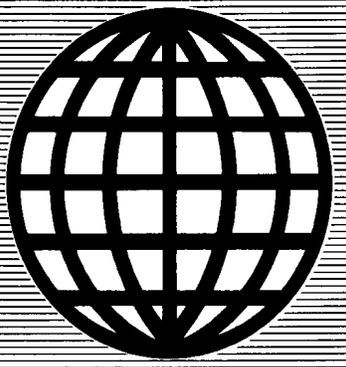


The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 50 May / June 1991 US$3.95

Offload a System CPU with Z181 Peripheral Control

Floppy Disk Alignment with the RTXEB and Forth

Motor Control with the F68HC11

Modula-2 and the Command Line

Home Heating & Lighting Control

Getting Started in Assemble Language

Local Area Networks

Z-System Corner

Using the ZCPR3 IOP

PMATE/ZMATE Macros

Z-Best Software

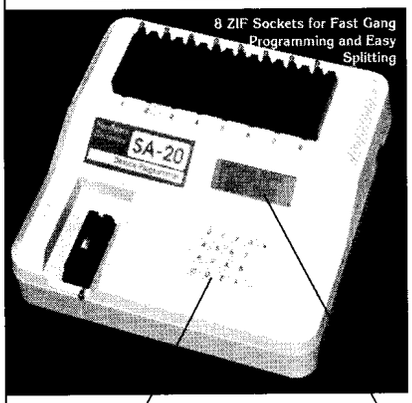
Real Computing

The Computer Corner

ISSN # 0748-9331

EPROM PROGRAMMERS

Stand-Alone Gang Programmer $750.00



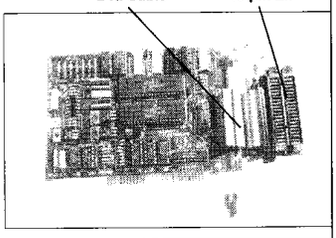
- Completely stand-alone or PC driven
- Programs E(E)PROMs
- ***1 Megabit of DRAM***
- ***User upgradable to 32 Megabit***
- ***.3/.6" ZIF socket, RS-232, Parallel In and Out***
- 32K internal Flash EEPROM for easy firmware upgrades
- ***Quick Pulse Algorithm (27256 in 5 sec, 1 Megabit in 17 sec.)***
- 2 year warranty
- Made in U.S.A.
- Technical support by phone
- Complete manual and schematic
- ***Single Socket Programmer also available. $550.00***
- Split and Shuffle 16 & 32 bit
- 100 User Definable Macros, 10 User Definable Configurations
- Intelligent Identifier
- Binary, Intel Hex, and Motorola S

20 Key Tactile Keypad (not membrane) **20 x 4 Line LCD Display**

Internal Programmer for PC $139.95

New Intelligent Averaging Algorithm. Programs 64A in 10 sec., 256 in 1 min., 1 Meg (27010, 011) in 2 min. 45 sec., 2 Meg (27C2001) in 5 min. Internal card with external 40 pin ZIF.

2 ft. Cable **40 pin ZIF**



- Reads, verifies, and programs 2716, 32, 32A, 64, 64A, 128, 128A, 256, 512, 513, 010, 011, 301, 27C2001, MCM 68764, 2532
- ***Automatically sets programming voltage***
- Load and save buffer to disk
- Binary, Intel Hex, and Motorola S formats
- ***Upgradable to 32 Meg EPROMs***
- ***No personality modules required***
- 1 year warranty • 10 day money back guarantee
- Adapters available for 8748, 49, 51, 751, 52, 55, TMS 7742, 27210, 57C1024, and memory cards
- Made in U.S.A.

NEEDHAM'S ELECTRONICS
4539 Orange Grove Ave. • Sacramento, CA 95841
Mon. - Fri. 8am - 5pm PST

Call for more information
(916) 924-8037
FAX (916) 972-9960

C.O.D.

Cross-Assemblers as low as $50.00
Simulators as low as $100.00
Cross-Disassemblers as low as $100.00
Developer Packages
as low as $200.00(a $50.00 Savings)

A New Project
Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

Get It To Market--FAST
Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

No Source!
A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you re-create the original assembly language source.

Set To Go
Buy our developer package and the next time your boss says "Get to work.", you'll be ready for anything.

Quality Solutions
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

BROAD RANGE OF SUPPORT
- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog Z80	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	Intel 80C196

- All products require an IBM PC or compatible.

So What Are You Waiting For? Call us:

PseudoCorp
Professional Development Products Group
716 Thimble Shoals Blvd, Suite E
Newport News, VA 23606
(804) 873-1947 **FAX: (804)873-2154**

William P Woodall • Software Specialist

Custom Software Solutions for Industry:
- Industrial Controls
- Operating Systems
- Image Processing
- Hardware Interfacing
- Proprietary Languages
- Component Lists

Custom Software Solutions for Business:
- Order Entry
- Warehouse Automation
- Inventory Control
- Wide Area Networks
- Point-of-Sale
- Accounting Systems
- Local Area Networks
- Telecommunications

Publishing Services:
- Desktop Systems
- Books
- CBT
- Format Conversions
- Directories
- Interactive Video

33 North Doughty Ave, Somerville, NJ 08876 • (908) 526-5980

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Chris McEwen

Technical Consultant
William P. Woodall

Contributing Editors
Bill Kibler
Tim McDonough
Frank Sergeant
Clem Pepper
Richard Rodman
Jay Sage

The Computer Journal is published six times a year by Socrates Press, P.O. Box 12, S. Plainfield, NJ 07080. (908) 755-6186

Opinions expressed in *The Computer Journal* are those of the respective authors and do not necessarily reflect those of the editorial staff or publisher.

Entire contents copyright © 1991 by *The Computer Journal* and respective authors. All rights reserved. Reproduction in any form prohibited without express written permission of the publisher.

Subscription rates• Within US: $18 one year (6 issues), $32 two years (12 issues). Foreign (surface rate): $24 one year, $44 two years. Foreign (airmail): $38 one year, $72 two years. All funds must be in U.S. dollars drawn on a U.S. bank.

Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 12, S. Plainfield, NJ 07080, telephone (908) 755–6186.

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used trademarks are acknowledged, and we apologize for any we have overlooked.

Apple II, II+, IIc, IIe, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder II, Dos Disk; Plu*Perfect Systems. Clipper, Nantucket; Nantucket, Inc. dBase, dBASE II, dBASE III, dBASE III Plus, dBASE IV; Ashton-Tate, Inc. MBASIC, MS-DOS, Windows, Word; MicroSoft. WordStar; MicroPro International. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C, Paradox; Borland International. HD64180; Hitachi America, Ltd. SB180; Micromint, Inc.

Where these and other terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

TCJ The Computer Journal

Issue Number 50 **May / June 1991**

Editor's Desk **2**

Offload a System CPU **3**
with Z181 Peripheral Control
By James J. Magill and Doug Woodburn.

Floppy Disk Alignment with the RTXEB and Forth **7**
Part Two
By Frank C. Sergeant.

Motor Control with the F68HC11 ..... **13**
By Matt Mercaldo.

Modula-2 and the Command Line **17**
Reading the ZCPR Command Line with Modula-2
By David L. Clarke.

Home Heating & Lighting Control **23**
Part Two, The Electrical Interface
By Jay Sage.

Getting Started in Assemble Language **27**
Making the Jump from High Level Languages, Part Two
By A. E. Hawley.

Local Area Networks **31**
By Wayne Sung.

Z-System Corner **33**
PCED, the Z-System for MS-DOS Computers
By Jay Sage.

Using the ZCPR3 IOP **37**
Add Function Keys to a Kaypro 10
By Lindsay Haisley.

PMATE/ZMATE Macros **41**
PMATE Facilities and Buffer-Saving Macros
By Clif Kinne.

Z-Best Software **45**
We're Off to the Libraries
By Bill Tishey.

Real Computing **47**
The 32FX16, CPU Caches and the Pi Benchmark
By Richard Rodman.

The Computer Corner **64**
By Bill Kibler.

Editor's Desk

By Chris McEwen

The lead article in this issue tells about a brand new 8 bit CPU from Zilog. When they asked if we wanted to play with a Z181 development system, I jumped. Sure! Art dearly loves running amok with new gadgets. From how they described this new chip, I knew it was right up Art's alley. But I baited Zilog. Why spend all that money making a new chip? The Z180 is a heck of a CPU. This elicited the same response you would expect of a new father: On board CTC! On board SIO! On board this-and-that-and-this-is-one-*hot*-chip! "Great," I said. "Tell us about it." Well, I've read the article and think they have something here. Take a look. If you come up with a good application for the Z181, let me know. I'd like to carry the article here.

The Silent Key

Before we go further, we owe a moment of silence to Irv Hoff. Irv's contribution to public domain programming was without measure. He co-authored BYE, the program we love to hate but without which no remote access CP/M system could run. But his work spans much more than that. It seems that I am constantly running some little tool that signs on with his name attached.

Irv passed away last month after a long illness. Please join me in a thought of thanks for his long and fruitful efforts on our behalf.

Trenton Computer Festival

We had a fine crowd in the CP/M conference at the Trenton festival. The presentations were great. Hal Bower is cooking up a banked operating system for systems over 64k that will give massive TPA. Well, it is massive for us 8-bitters, you understand. Remember, this is the side of the room where we count bytes, not megabytes.

Bruce Morgen gave a discourse in programming for Z-System. Al Hawley's series of articles on Assembling Language programming is quite timely in this regard and from the crowd's reaction, he hits the nail squarely on the head. By the way, Al was there to hear the praise. No amount of money can reward one as well as compliments and Al was quite well paid that day. Deservedly so, I might add.

Humility makes it hard, but I should acknowledge similar payment. As we went around the room introducing ourselves, Jay Sage would mention what each person was doing. When my turn came, he mentioned that I published *TCJ*. There was resounding applause. If you were there, I thank you. But the reality is that the applause belongs to the authors—they're the ones who make this journal so unique.

It was a very long day, continuing on well into the night. Steve Dresser and I wrapped it up about 1:30 in the morning and left Jay, Al Hawley, Howard Goldstein, Bob Dean, and Ian Cottrell to explain the loud party to the hotel management. What can I say? This was all serious work, you know.

Can You Say "Oops"?

In the last issue, I told how Mark Burrows will be writing about using Fidonet on a CP/M box. This was news to a great many, most especially Mark Burrows. My apologies. I've been calling several Texas systems, including Mark's. But the real name behind the story is Marc Newman! Marc (not Mark) wrote MSBBS in Turbo Modula-2. That's the Modula-2 compiler that never existed, remember? Well, I've been popping in on Marc's system for the last month or so, and it's the best vaporware I've ever seen. Fact is, he has it working like a charm.

Marc distributes his system in a new way. It is not public domain, though he gives you full source code. And it is not shareware as he asks nothing for himself. He calls it "charityware," and asks you to make a donation for whatever you feel is right to MS. I like that. Anyway, I am hoping to see something on this in the next couple of months.

Another topic begging discussion is Usenet on CP/M. I still have a marker out there. Let's see how it goes. These are important topics as they help bring people together. It is exhilarating to engage in electronic conversation with folks around the world.

Home Control, X10 and More.

Jay continues his series on home control in this issue. The more he writes, the more intriguing it becomes. A user on Jay's Z-Node says that he is putting in the necessary wiring as he builds his house this summer. Meanwhile, Rick Swenton has pulled his partner, Biff Bueffel, into writing on X10. These two have been working together on programs for X10 and I expect great things.

Roger Warren pulled rank on me. He wants to write about modifying the Ampro to use a USR Dual Standard modem. I wanted an article on his backup scheme. After reading his article (it will be in the next issue), I humbly defer to Roger's judgment. You won't be disappointed.

We have a new author in this issue. Lindsay Haisley heard that I was looking for an article on programming IOPs and offered his work on remapping the Kaypro keyboard. This is good information right out of the box if you own a Kaypro, but stands as fertile ground for developing IOPs for other machines and purposes. You will enjoy Lindsay's write-up.

We also have a new editor! Frank Sergeant has accepted the position of Forth Editor. Frank's article on the Floppy Disk Aligner won first place in the Harris Semiconductor RTX contest last year and we are honored to be carrying it in our pages. I will defer to Frank's better judgement when it comes to Forth.

See Editor, page 56

Offload A System CPU

with Z181 Peripheral Control

By James J. Magill and Doug Woodburn

Introduction

To remain competitive, companies are in a constant state of flux developing new, higher performance products that offer greater system performance and functionality, all at reduced system level costs. This needs to be done in a timely manner to meet certain market windows, with minimal risk to the schedule. As you might well expect, this is quite a challenge for both software and hardware designers.

To feed this insatiable appetite for new products with a minimum design cycle, designers generally opt for migration from an existing hardware and software platform (*Figure 1*). Thus, computer architectures are trending towards distributed processing where intelligence is put in the peripheral function. This offloads the central processor from the I/O tasks, thereby increasing the overall system performance. Further, it tends to make the architecture modular in nature allowing you a much easier way to modify for later designs.

System level performance can be approached in several ways. For example, a higher speed system clock can be used. A 20 MHZ Z80 CPU is available today, but higher speed brings with it noise in the form of EMI. It makes no sense to increase the performance at the expense of having to add extra cost to the system in the form of screening ferrite beads, etc. Another approach is that of increased microprocessor bandwidth. However, one should think real hard before deciding to utilize a 16-bit processor. To make full use of a 16-bit architecture, perhaps one should use a 16-bit peripheral, memories, et cetera. Also, one must not lose sight of the final cost objectives of the system.

A third and popular approach is to utilize the advances in 8-bit architectures. Say, for instance, a device was available that could provide an increase of 25% in performance for the same clock speed, while running the same code. Now, you can attain many of the previously mentioned objectives. The device, the meat of this article, is a highly integrated general purpose intelligent peripheral or controller, that is used in a wide variety of applications including data communications, terminals, programmable controllers, printers, and modems.

Let us examine a typical peripheral within an overall system context (*Figures 2 and 3*). Such a peripheral requires intelligence, in the form of a CPU, to control the peripheral function. The advent of larger programs provides MMUs that can prove useful in extending the addressing capability. The CPU on the Z181 SAC (Smart Access Controller - *Figure 4*) is an enhanced version of the Z80 which provides an approximate 25% increase in performance from the same clock speed, with additional instructions.

A peripheral typically communicates with a host computer or network to transfer information or to receive downloaded commands. Hence, a multi-protocol synchronous serial communications channel is required to provide the fast data transfer. Often a DMA channel is assigned to each receiver and transmitter channel of the serial I/O device to offload the CPU and complete transfers without CPU intervention. Given that generally the communication is at the local peripheral, where the communications are generally asynchronous, the SAC provides two UARTs with on-board baud rate generators.

> "It is worthy to note that software siphons off most of the development dollars."

Parallel I/O is often required for a printer, display keyboard, et cetera. The SAC provides 16 I/O channels. In order to ensure design flexibility of a highly integrated design system, designers need a wealth of counter timing functions. The SAC provides 4x8-bit counter timers in the on-board CTC (Z84C30) megacell together with two 16-bit timers. Of course, this all needs to be done within the cost constraints of the project. With Superintegration ™, the functionality of the above can be accomplished within one 100-pin quad flat pack device, no bigger than one's thumb nail (*Figure 4*) . Although most of the foregoing addresses the

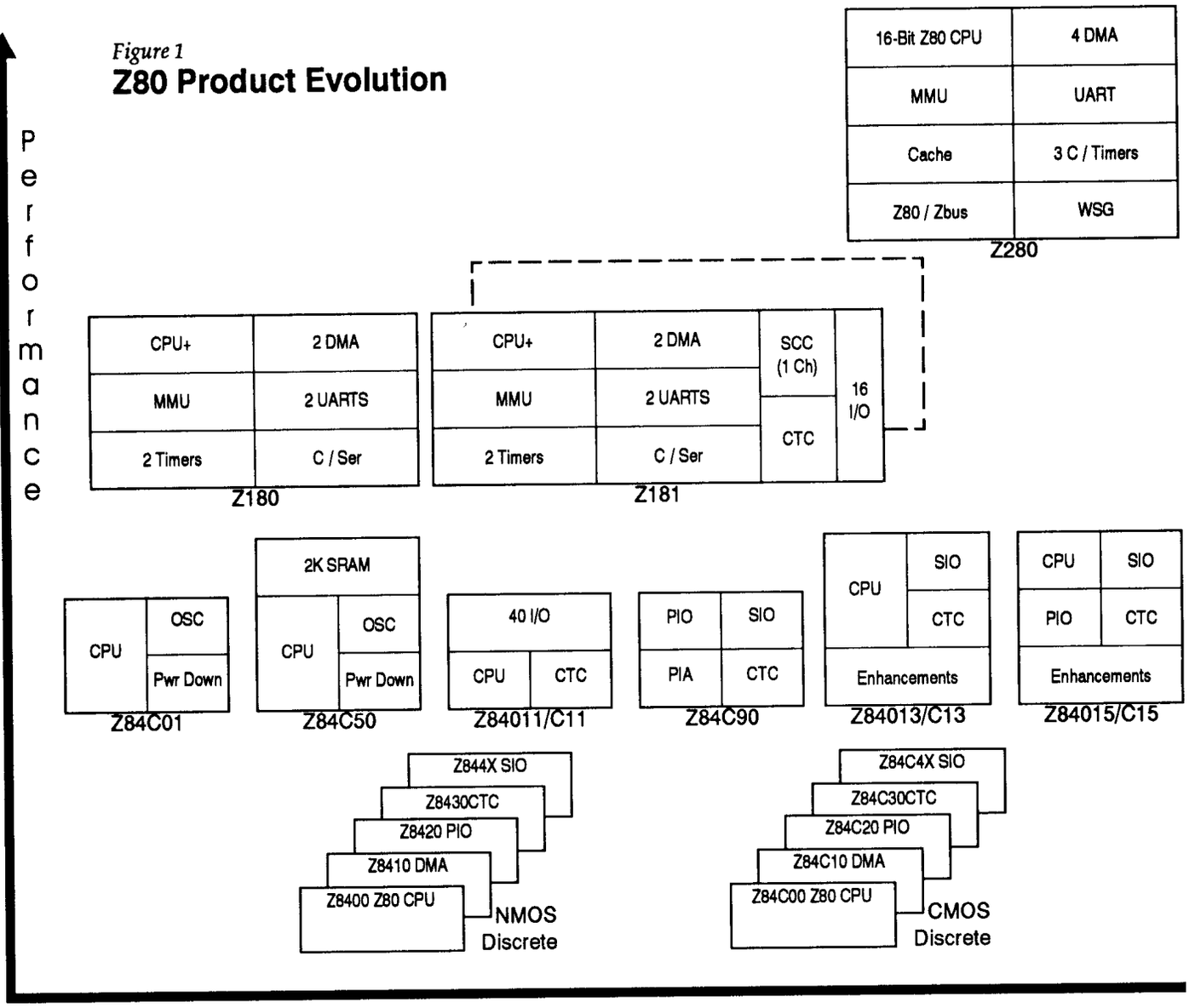
James Magill is director of the Intelligent Peripheral Product Line at Zilog. He started his career at the British Broadcasting Corporation in London as a Senior Engineer, then joined Texas Instruments in their consumer and logic products manufacturing area. Prior to joining Zilog, Mr. Magill spent 10 years at Signetics holding Product Control, Test Area Manager and Marketing Manager positions in their microprocessor division.

Mr. Magill holds a Masters of Science in Operations Management from City University, London and a Bachelors of Science with honors in Electrical Engineering from Queens University, Belfast, Northern Ireland.

Doug Woodburn received his B.S. degree from the University of California Long Beach in Industrial Technology with a major in Electronics. He has worked as a Product Engineer for NCR Electronics Division and for Varian Data Machines, both in computers. He switched over to a position as a Senior Technical Writer at Varian in 1970 and has remained one for the last 21 years. Doug joined Zilog nine years ago and has published seven media articles in the last 18 months. Prior to this, Doug wrote hardware manuals for Zilog's System 8000 series.

Figure 1

Z80 Product Evolution



hardware aspects of the design, it is worthy to note that software siphons off most of the development dollars. The SAC already has the software built in. Furthermore, in the peripheral control function where such attributes as fast interrupt response time become paramount, the software is written in assembly as opposed to high level languages, to speed the response. Sometimes both high level languages and assembly language are mixed with the assembly language being used for fast control and timing loops. This overcomes the inherent inefficiencies of higher level languages. The Z80 product portfolio outlined in Figure 1 provides a wide range of code compatible devices offering differing functionality and performance levels.

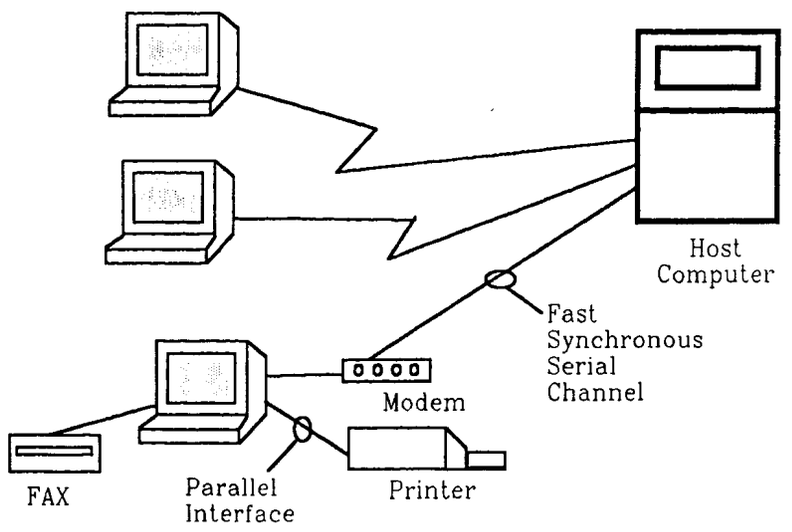
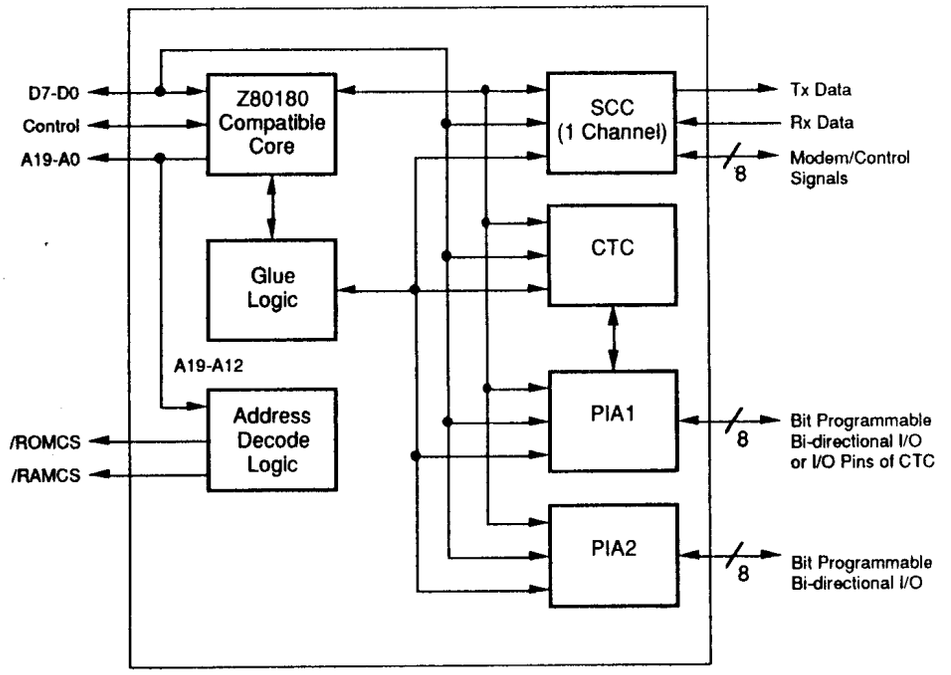


Figure 2 **Typical Embedded Controller Environment**

Synchronous Talk with the System CPU

Embedded Control gives you cost performance applications by using 8-bit vs larger 16 or 32-bit intelligent processors (Figure 5). When you want to talk to the System CPU, the SAC provides one fast synchronous transmit/receive 8-bit channel via its on-

Figure 3 **Z80181 Block Diagram**



Z80181 = Z180 + SCC + CTC + PIA

chip SCC with a serial transmission rate of 2.5 Mbits per second (Figure 6). The SCC logic unit provides you with a multi-protocol serial I/O channel. Its basic functions, like serial-to-parallel and parallel-to-serial conversions, can be programmed by the MPU for a broad range of serial communications applications. This logic unit supports all common asynchronous and synchronous protocols; Monosync, Bisync, SDLC/HDLC, byte or bit oriented.

The PCLK for the SCC is connected to PHI (System clock), the /INT signal is connected to /INT0 internally (requires external pull-up resistor) and the SCC is reset when the /RESET input becomes active. Interrupt from the SCC is handled via Mode 2 interrupt. During the interrupt acknowledge cycle, the on-chip SCC interface circuit inserts two wait states automatically. These two wait states buy you access settle time with the On-Chip Timer (Figure 7).

The on-chip CTC provides you with four separate 8-bit Counter/Timer channels (Figure 8). They are programmed by the MPU for a broad range of counting and timing applications. You can use them in the areas of event counting, interrupt and interval counting, and serial baud rate clock generation.

Each of the designated CTC channels 0-3 have 8-bit prescalers (when used in timer mode). Also, each one has its own 8-bit counter to provide a wide range of count resolution. Each of the channels have their own Clock/Trigger input to quantify the counting process and an output to indicate zero crossing/timeout conditions. These signals are multiplexed with the Parallel Interface Adapter 1 (PIA1). With only one interrupt vector programmed into the logic unit, each channel generates a unique interrupt vector in response to the interrupt acknowledge cycle.

Parallel and Asynchronous Communication

The SAC has two 8-bit PIA Ports called PIA1 and PIA2. Each port has two associated control registers; a Data Register and a bit direction register (input or output). PIA1 is multiplexed with the CTC I/O pins. When you select the CTC I/O feature, the CTC I/O functions override the PIA1 feature. Mode selection is made through the System Configuration Register (Ad-

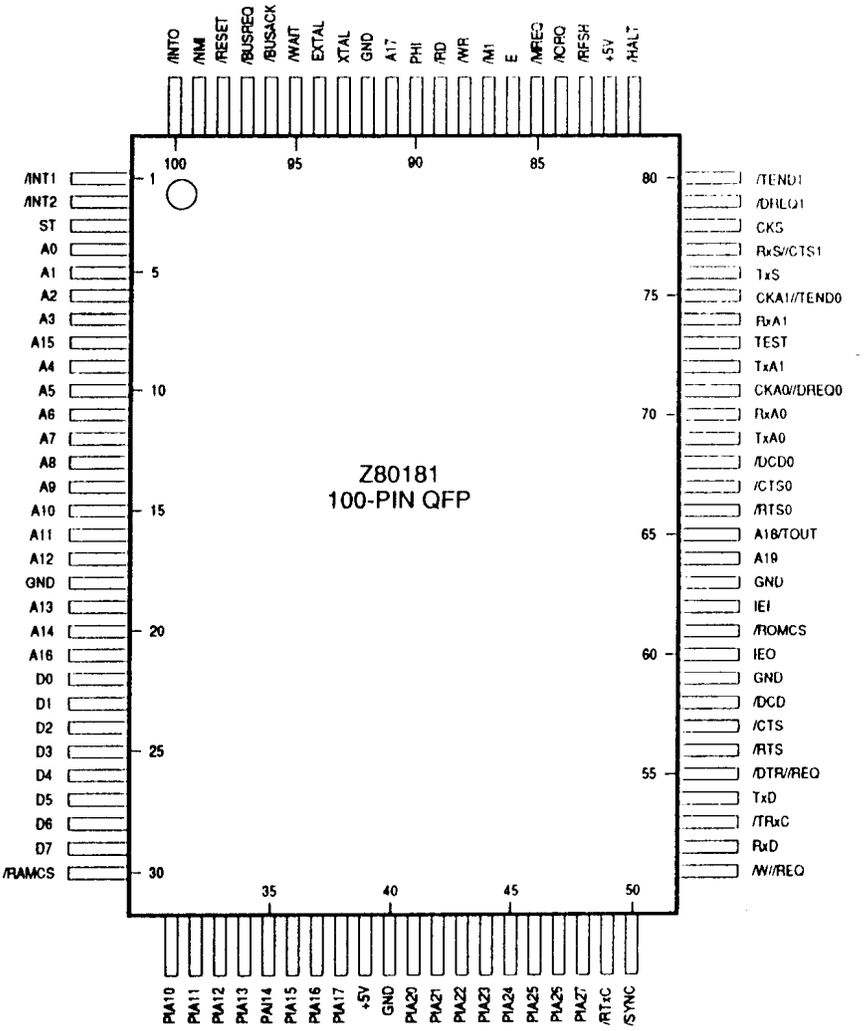
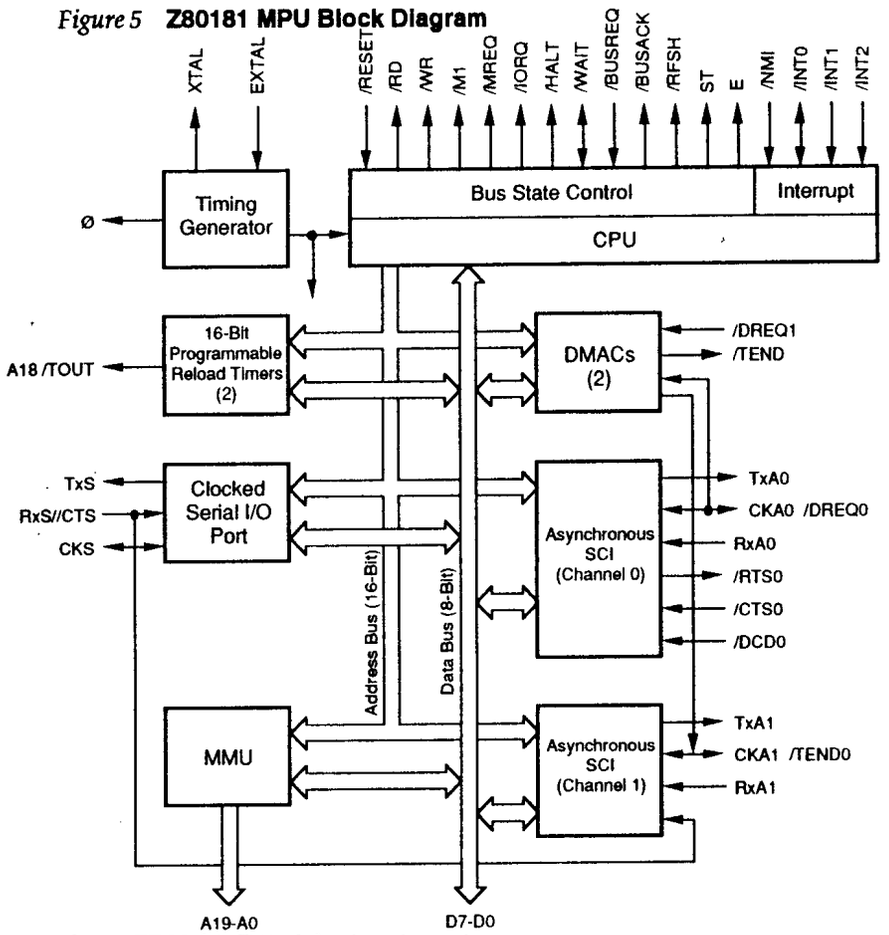


Figure 4 **Z80181 Pin-out Assignment**

Figure 5 **Z80181 MPU Block Diagram**



dress: EDH; bit D0). Schmitt-trigger inputs were designed into the PIA1 to provide a better noise margin. After reset, these ports become inputs.

You have your choice of various protocols with which to interface and then down load data; You'll find two other asynchronous channels on the SAC for communications between terminal and printer, terminal and keyboard, terminal and scanner, etc. Being able to work with these two asynchronous ports just gives you that much more flexibility.

Interrupt Structure

The Z80 architecture, with it's dual register bank architecture, is well suited to embedded control applications by providing you with a fast interrupt response time or context switching. If you want to service one channel while setting up the other, the SAC has a complete context switching capability. This dual register bank switching approach halves the latency switching time.

To control interrupts, the Interrupt Enable In (IEI) signal is used with Interrupt Enable Out (IEO) to form a priority daisy chain when there is more than one interrupt driven peripheral. IEO controls the interrupt of external peripherals. You'll find it active when IEI is a 1 and the MPU is not servicing an interrupt from the on-chip peripherals.

Note: Reference Figures 7 and 9 for help in the explanation of the following text:

One way of connecting an external peripheral to the SAC is as an input device when it has higher priority than on-chip peripherals (Figure 9A). The second way is when you want the external peripheral to have a lower priority than the on-c hip peripherals (Figure 9B). When the external peripheral is a higher priority, the IEI-IEO delay has to be less than two clock cycles.

In order to meet the on-chip SCC and CTC timing requirements, the SAC interface logic inserts another three wait states into the interrupt acknowledge cycle. (You'll find a total of five wait states; includes two automatically inserted wait states.) Therefore, to meet the timing requirements, the SAC's on-chip circuits generate interface signals for the on-chip SCC and CTC.

To give you a better understanding of the external vs internal peripheral interrupts in relation to the SAC and the daisy chain arrangement, reference the three following cases in relation to Figure 7.

Case 1 - On-Chip SCC: The SCC/INTACK goes active on the T1 clock fall time. The settle time is from SCC/INTACK active until the SCC /RD signal goes active on the fourth rising wait state clock.

Case 2 - ON-Chip CTC: The settle time for the on-chip /IORQ is between the fall of /M1 until the internal CTC / IORQ goes active on the rise of the fourth wait state (the same time as SCC /RD goes active).

Case 3 - Off-Chip Z80/Z180 peripheral: The settle time for the off-chip Z80/Z180 peripheral is from the fall of /M1 until CTC /IORQ goes active. Since the SAC's external IORQ signal goes active on the clock fall of the first automatically inserted wait state (T_{WA}), the external daisy-chain device has to be connected to the upper chain location. Also, it must settle within two clock cycles.

See Zilog, page 59

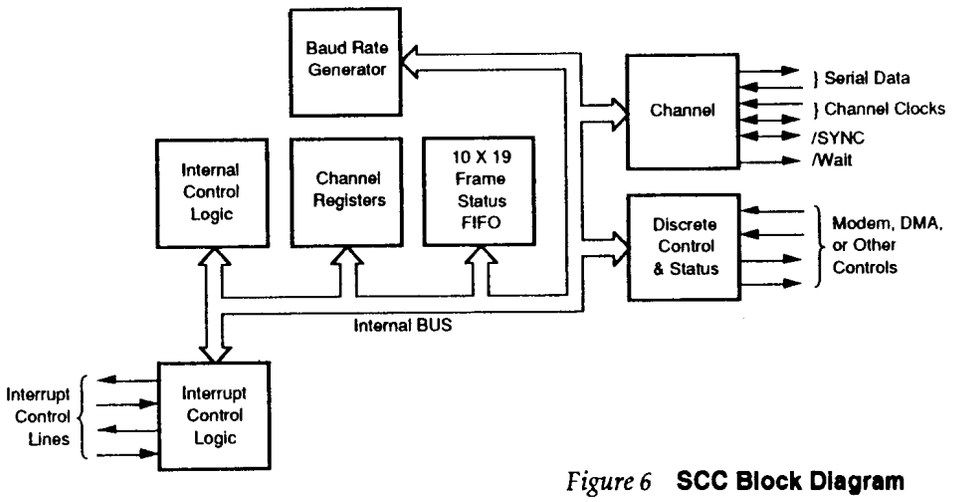


Figure 6 **SCC Block Diagram**

Floppy Disk Alignment with the RTXEB and Forth

Second of Three Parts

By Frank C. Sergeant

We continue now with Frank Sergeant's first place winning entry in the Harris RTX design contest—Editor

Disk Drive Control Signals

Five outputs are used to send control signals to the drive: Direction, Step, Write Data, Write Enable, and Side Select. These are all digital signals. The *Motor On and the drive select control signals could be connected to outputs, but are not; they are hardwired active.

Signal Conditioner

Only one analog signal is needed from the drive: from the head read amplifier. This is picked up with a test lead or alligator clip (from TP1 or TP2 on Tandon drives). After conditioning and comparing, this will trigger an interrupt on EI3.

The signal we need to condition comes from the output of (one of) the differential head read amplifiers. It is an AC signal with about a 300 to 500 mV swing on top of about 6 volts DC. We run it through a capacitor to get rid of the DC and through an op amp to level shift so none of the signal goes below ground (we put back a little DC). That's all. This lets us use a single power supply op amp. We could go to greater complexity and use both of the differential signals (TP1 and TP2) but it isn't necessary in this application. This conditioned signal will usually be referred to as 'the signal.' It is *not* the raw read (digital) signal from the 34 pin disk cable.

Digital to Analog Converter (DAC)

A simple R-2R resistor network connected to the upper 8 output bits does all the work. It makes a voltage divider with 256 possible settings between 0 and 5 volts. The output bits, being CMOS, connect their ends of the resistors to either GND or Vcc. This is not the best range to match to the conditioned input signal, so it is scaled downward by a fixed voltage divider, to give a range between 0 and 450 mV. The word DAC! sets the level (and leaves the other bits of the output port unchanged). 0 DAC! sets it to the minimum level and 255 DAC! sets it to its maximum. Currently the prototype uses 47K ohm SIP resistor packs. A resistor is used alone to make a '2R' and two are paralleled to make an 'R'. This is fast and works great but uses more pins than necessary. I might use an integrated DAC for production.

The DAC is used to set a reference voltage that will be compared with the signal from the disk. This reference will be referred to as Vref.

The DAC is used by the software as part of the analog to digital converter (ADC), in order to measure the analog signal from the drive.

Comparator

The conditioned signal (hereafter just called 'the signal') is connected to the non-inverting input of the comparator. Vref (from the DAC) is connected to the inverting input of the comparator. As long as the signal is lower than Vref, the comparator will be low. If the signal goes above Vref, even briefly (as at the top of a wave-form), the comparator will go high and trigger an interrupt. The inputs were connected this way (Vref to the inverting input), so that a high would mean the signal exceeded Vref, because the interrupt (EI3) is active high.

The Aligner—Software Functions

Status

The Aligner reads disk drive status signals through the input port. As described under the section on I/O Ports, this is done with the phrase 31 G@ (see screen #3606). We then AND this value with a bit-mask to isolate the bit we are interested in. Since these status lines are active low, a resulting zero means the line was active and a non-zero means the line was inactive. We usually convert this to positive logic with the word 0=.

Frank Sergeant is a hardware/software consultant specializing in business and/or realtime systems. He is the author/implementor of Pygmy Forth for PC/MS-DOS systems (version 1.3 is available from FIG, GEnie, and fine BBSs and shareware houses everywhere). He has been designing, building, and programming microcomputer systems since the late '70s. One of his greatest joys is replacing hardware with software. He is in the process of porting Pygmy to the Super-8, 68HC11, RTX, etc. His floppy disk drive aligner entry won the RTX design contest. Shortly thereafter he was shocked to hear the RTX was being abandoned by Harris. However, recent conversations with Harris officials have reassured him that it was only future development that was abandoned. Harris has fully, publicly committed to producing the RTX for a minimum of 2-1/2 years. In light of that, Frank breathed a sigh of relief and continues his RTX development work. Frank can be reached as F.SERGEANT on GEnie or through TCJ.

```
scr # 3611
( DAC!  write to 8 bit R-2R DAC )

: DAC! ( u -)
  7 FOR 2* NEXT ( u*256)
  OUT @ 255 AND     OR     P! ;
  ( put it in high half of output port; don't change low half)

(    note that we do not save the dac value in OUT as we do  )
( for the disk drive control bits - the lower byte remains   )
( undisturbed and the lower byte of OUT is still valid.      )
```

scr # 3911
Digital to Analog Conversion

This system uses an 8 bit DAC built with an R-2R voltage divider network, using 47K ohm SIP resistor packs. 2R equals 47K and R equals half of that (2 47K resistors in parallel). This gives a 256-step output between GND and the power supply. This is further scaled by a fixed voltage divider (100K & 10K) to cut the output by about an eleventh (a potentiometer could be substituted), giving a closer match to the signal to be measured.

A similar result can be achieved by using an integrated 8-bit DAC.

DAC! Set the output voltage level.

```
scr # 3612
( EI3-INTW          interrupt handler used by CLOCK )
           ( save time when each peak starts, starting at PAD)
      ( the beginning of each peak will trigger this interrupt)

: EI3-INTW ( -)
  TC1@  CR@  ( time cc)
  1 #PEAKS +!
  IMR@ ( time cc imr)
  EI3 NOT IMR!            ( ie mask all interrupts except ei3)
  ROT PAD #PEAKS @ 2* + ( cc imr time a) !   ( cc imr)
  BEGIN    CR@ 0>             (    ie msbit of CR will be set)
                              ( as long as comparator is high)
     TC1@ 1000 U<  ( ie bail out if timer counts down too far)
  OR UNTIL                    ( loop until wave peak goes away)
  IMR! CR!   ;
```

scr # 3912
EI3 interrupt routine to map windows for azimuth bursts

EI3-INTW ("W" for "window") This stores the current timer value at PAD and waits for the comparator to go low again, so we don't count a wave more than once. These timer values represet the count-down values from the start of the index pulse. All of the peaks of interest occur in about the 1st two milli-seconds.

This int handler just stores the values. It is set up by CLOCK. Then MARK is used to figure out from these times where the azimuth bursts occur.

```
scr # 3613
( CLOCK  find when azimuth bursts occur )
: CLOCK  ( - )
  Vb @ 4 + DAC!     ( set dac so any wave peak will trigger int)
  PAD 200 ERASE     ( clear at least the 1st few bytes for data)
  ['] EI3-INTW  10 !INTERRUPT       ( install interrupt handler)
  0 #PEAKS !                          ( initialize wave counter)
  SYNC                                  ( wait for index pulse)
     0 TC1! EI3 UNMASK       ( start timer and allow interrupts)
  2 MS                      ( 2 milli-seconds is plenty of time)
  EI3 MASK   ;                          ( stop interrupt handler)
```

scr # 3913
Find when azimuth bursts occur

CLOCK Set DAC to a value low enough that every wave peak should trigger the comparator. Set up the EI3 interrupt handler. Wait for the index pulse (SYNC). Initialize timer1. Note that timer1 is not enabled as an interrupt source, but that the EI3 interrupt (connected to the comparator's output) reads timer1 to find out when the current wave peak occurs. We let it run for "2 MS" (actually longer because of the time spent in the interrupt handler) and then close down the operation by masking the interrupt.

The data we have gathered will be processed by MARK.

```
scr # 3614
( EI3-INT  note it if comparator ever goes active )
: EI3-INT ( -)
  -1 COMP-FLAG !     EI3 MASK  ;

( this is just about the ideal size for an interrupt handler)
```

scr # 3914
Note whether the comparator ever goes active.
EI3-INT This is a very simple interrupt handler. If it is invoked (by the comparator going active) it notes that fact by setting COMP-FLAG true. Then, it disables itself. It is used to let us know if the disk drive signal exceeded the voltage reference level set by the DAC at any time during a particular time interval. It turns itself off so it won't be invoked repeatedly during the high part of a wave.

On the RTX, unlike most processors, the return from interrupt instruction is the same as a return from subroutine. This allows an interrupt handler to be tested from the keyboard just like any other Forth word!

The word INX? is true during the index pulse. The following definitions (from screen #3605) allow us to synchronize with the disk:

```
: P@     ( - u)  31 G@  ;  MACRO
: INX?   ( - f)  P@ 1 AND 0=  ; ( true during index pulse)
: ?INX     ( -)  BEGIN   INX?     UNTIL  ;
: ?NOT-INX ( -)  BEGIN   INX? 0=  UNTIL  ;
: SYNC     ( -)  ?NOT-INX   ?INX   ;
( possibly wait through the current index pulse before)
( looking for the next one )
```

The word ?INX waits until the index line is low. The word ?NOT-INX waits until the index line is not low. Note the convention of putting the question mark on the end of words that return a flag and putting the question mark on the beginning of words that do something that is conditional in some way (but that probably don't return a flag). Thus, ?INX does something conditional: it hangs in a loop until INX? returns a true flag.

Both ?INX & ?NOT-INX are combined into the word SYNC which guarantees to wait until the next index pulse starts. At first glance you might think that ?INX would accomplish this on its own, but it would fail if executed while the index line was already low. It would immediately return because INX? would be true, but we wouldn't know that we were at the exact beginning of the index pulse.

The status lines for *Write-Protect and *Track0 are read similarly.

Control

The disk lines to control the motor and to select drive 0 and drive 1 are hardwired active (low) to reduce the number of output lines we need. This is not really necessary, as we have some spare output lines, but it does remove one more complexity from the program. Usually you would only want to connect one drive to the Aligner at a time. This way, whichever drive you connect, it will be selected and its motor will always run. Drives can be jumpered to respond either as drive 0, 1, 2, or 3, but 0 and 1 are the most common. As long as it is jumpered as 0 or 1, you do not need to do anything special to select it for the Aligner.

Screen #3604 shows the bit-masks for the control lines. These are used just as in the LED example. This application does not use the Write or the Write Enable lines, but we make sure they are set inactive (high) when we initialize the output port to all ones. An extension to this project would include using those lines for a write test.

That leaves only 3 control lines to handle: HEAD (Side Select), *DIR (Direction), and *STEP. HEAD is used to select which of the two heads will be active. The following words from screen #3607 do this:

```
H0 ( -)  HEAD OFF ;
H1 ( -)  HEAD ON  ;
```

The *DIR line determines whether the head will move toward the innermost track (higher track numbers) or toward the outermost track (lower track numbers) whenever the *STEP line is pulsed.

The variable TRK holds the value of the current track so we'll know which direction and how far to step in order to move to a specific track. For this to work, TRK must be initialized to the actual track the head is on. The only way to achieve this initial synchronization is to step the head outward until the track 0 switch goes true. At that point, the head is on track zero and we store a zero into TRK. Thereafter, each time we move the head, we update TRK. See screen #3608.

DAC: Setting Voltages

The 8 bit DAC determines a reference voltage (Vref). We set Vref by writing a value between 0 and 255 to the upper output port connected to the DAC. The only caution is that we must not disturb any of the lower 8 bits. DAC! from screen #3611 takes care of this by shifting the DAC value 8 bits to the left. We could do it with 256 * or -8 SHIFT but the phrase 7 FOR 2* NEXT does it faster. This value is then combined with the value of the lower 8 bits, as follows:

```
: DAC!  ( u -)  ( u is between 0 & 255)
   7 FOR 2* NEXT ( u*256)
   OUT @ 255 AND  ( isolate lower 8 bits)
   OR  ( combine with DAC value)
   P!  ;
```

We do not have to update OUT as long as we don't change the lower 8 bits.

Examples of setting voltages with DAC!:

```
255 DAC!     ( set Vref to its maximum )
127 DAC!     ( set Vref to mid-scale   )
  0 DAC!     ( set Vref to its lowest  )
```

Whenever the input signal is higher than the reference voltage the output of the comparator will be high.

THE COMPARATOR INTERRUPT ROUTINE

For measuring peak amplitudes of the signal, all we want the interrupt to do is set a flag true and disable itself. We disable the interrupt for two reasons: (1) once is enough to tell us the peak exceeded Vref and (2) the interrupt is level sensitive. That is, the interrupt would be continuously active until the wave-form went below Vref and we have other things to do than hang in the interrupt handler. Here's the comparator interrupt handler:

```
: EI3-INT  ( -)   -1 COMP-FLAG !    EI3 MASK    ;
```

It is installed from within another word with the phrase

```
[']  EI3-INT 10 !INTERRUPT
```

This shows how simple interrupts are to deal with using Forth on the RTX. To use the interrupt, set the DAC, clear COMP-FLAG, wait a little while, and check COMP-FLAG. If COMP-FLAG is now non-zero we know the comparator has been triggered.

Sometimes we use the DAC & comparator for another purpose: to time the wave-form rather than measure its amplitude. We set the DAC just a little above the signal's quiescent level so every wave will trigger the interrupt. We time the period between interrupts. This uses a slightly different interrupt handler that waits for the comparator to go low again before returning. That way we count each peak only once.

ADC: Measuring Voltages

All the tests the Aligner performs measure either the time between events or the amplitude of the analog read signal. Measuring time is easy with the speed of the RTX and its

```
scr #  3615
( ADC  successive approximation analog to digital conversion )
: ADC ( - c)     ( time delay must be stored in SR )
  ['] EI3-INT  10 !INTERRUPT
  0 255 127  ( lo hi mid) DUP DAC!  50 CYCLES
  7 FOR  ( lo hi mid)
         0 COMP-FLAG !
         EI3 UNMASK  SR@ CYCLES  SWAP ( lo mid hi)
  COMP-FLAG @
        IF ( dac is too low) ROT THEN DROP
        ( lo hi) 2DUP + 2/ ( lo hi mid) DUP DAC!
  NEXT  EI3 MASK  >R  2DROP R>  ;
```

scr # 3915
Successive Approximation Analog to Digital Conversion

ADC This measures the peak voltage of a waveform. The signal to be measured is one input of a comparator. The voltage reference set by our DAC is the other input. Whenever the signal to be measured exceeds the DAC, the comparator output goes high, triggering the EI3 int handler, which sets COMP-FLAG. We have 8 bits to set so we go thru the loop 8 times. Each time we split the difference between our high & low boundaries and wait a while (determined by the value previously loaded into SR) and see whether we were were too high or too low. This midpoint then becomes either the new high or the new low boundary for the next pass.

```
scr #  3616
( !Vb  measure quiescent voltage level when there's no signal)

( this establishes a base level to compare azimuth burst)
( amplitudes to)

: !Vb ( -)
  8 SR!         ( set up the delay used by ADC)
  SYNC          ( wait for the index pulse)
  ADC           ( quick, measure the voltage before data starts)
  Vb !  ;       ( save the base voltage reading)
```

scr # 3916
!Vb Use the ADC to find the quiescent voltage of the azimuth track. We've got to be quick!

```
scr #  3617
( These words help analyze the data collected by CLOCK )

: Tpk ( pos - u) 2* PAD + DUP @ SWAP 2- @ SWAP- ;
  ( this is elapsed time for this peak since previous peak)
  ( "tee peak" )

: Tabs ( pos - timer-value ) 2* PAD + @  ;
  ( this is elapsed time since index pulse)
  ( "tee abs")
```

scr # 3917
Tools to help analyze the timing data stored at PAD by CLOCK

Tpk ("tee peak") This returns the time in RTX clock cycles from the previous wave peak to the current wave peak. The actual down-counting timer value from the index pulse is what is stored at PAD. Since the current reading is compared to the previous reading, we have a zero value dummy position in the 1st slot.

Tabs ("tee abs") This is elapsed time since index pulse for the current wave peak.

```
scr #  3618
( More help in analyzing the data collected by CLOCK)

: PEAK<?  ( pos value - pos+1 f)
  ( "peak less than"    think of it as similar to  0<  )
  ( true if value is less than peak or if at end of data )
  OVER Tpk U< ( pos f) OVER SR@ < 0= OR  SWAP 1+ SWAP ;

: <PEAK<? ( pos lo hi - pos+1 f)
  ( "peak within")
  ( true if peak is within lo & hi or if at end of data )
  2>R DUP 1+ SWAP DUP Tpk 2R> WITHIN SWAP SR@ < 0= OR  ;
```

scr # 3918
Tools to help analyze the timing data stored at PAD by CLOCK

PEAK<? ("peak less than") Think of it as similar to 0<. This tells us if the current peak interval is greater than the number on top of the stack. It also returns true if there are no more peak intervals to look at. (True if value is less than peak or if at end of data.)

<PEAK<? ("peak within") This is true if peak is within lo & hi or if we are at end of the data.

These two words also increment the position. I'm on the look out for better names for these or better factoring, but at least they are easy to test and they help make the next definition more readable (see MARK).

```
scr #  3619
( Locate azimuth burst windows by reading times at PAD )
: MARK ( -)
  WINDOWS 16 ERASE  #PEAKS @ ( end-marker) SR! (  )
 1 BEGIN     500 PEAK<?  UNTIL  ( index to data burst)
   BEGIN    1000 PEAK<?  UNTIL  ( data burst to 1st az burst)
  WINDOWS  SWAP 3 FOR  ( a pos)
   BEGIN  30 150 <PEAK<? UNTIL
          2DUP 1- Tabs SWAP ! SWAP 2+ SWAP
   BEGIN      200 PEAK<? UNTIL
          2DUP 2- Tabs SWAP ! SWAP 2+ SWAP
                 NEXT ( a pos)  2DROP  ;
```

scr # 3919
MARK This studies the times of the wave peaks on the azimuth track to figure out when the azimuth bursts occur. The data was placed at PAD by CLOCK (using EI3-INTW). MARK leaves the timer values for the azimuth windows in the WINDOWS array.

This info is used by the azimuth tests. There should be a gap of at least 500 (1200-1300 typical) clocks from the index pulse. Then there is a data burst. Then there is another gap of at least 1000 clocks (2300 perhaps). Then there are 4 groups of about 61 125 KHz bursts. These should have a period of about 64 clocks, but we'll accept anything "close" (within 30 to 150). These timings are used by AWAIT & AZ so that the azimuth burst peak amplitudes can be read at the correct times.
scr # 3920

three built-in timers. That leaves the problem of measuring voltage. The RTX does this by setting the DAC to a certain level and checking the comparator to see whether the signal exceeds that level. Software controls this process, converting the DAC and comparator into an ADC. This not only saves the cost of an ADC but simplifies the conditioning circuitry needed to measure the peaks of the wave-forms. Again, the speed of the RTX helps eliminate hardware.

The 8 bit DAC determines a reference voltage (Vref). Set Vref by writing a value between 0 and 255 to the output port connected to the DAC using the word DAC!. Whenever the input signal is higher than the reference voltage the output of the comparator goes high. The comparator is connected directly to external interrupt pin EI3. If we set Vref higher than the peak amplitude of the input signal the comparator will never go high and the interrupt will never be triggered.

The level of the input peaks can be determined by setting Vref to various points and seeing which points cause an interrupt. We start looking within a range of 0 to 255 (the maximum range of values possible with an 8 bit DAC). We set the DAC to the middle of the range (to 127) and wait a little while and see if the interrupt was triggered. If it was, we know that the peak exceeded Vref, so we narrow the range to 127 to 255. Again we split the difference and set the DAC to the middle of the (new) range (to 191), wait, and see if the interrupt was triggered this time. If it was, we narrow the range to 191 to 255. If it wasn't, we narrow the range to 127 to 191. We continue this process until the DAC has zeroed in on the answer. It only takes 8 times to find it with an 8 bit DAC. This method of analog to digital conversion is called the successive approximation method. Programmers call it a binary search.

Note that we not only have a software ADC, we also have a software peak detector, without having to build a peak detector circuit as part of the signal conditioner.

By taking these readings on a regular basis and plotting them we have a digital oscilloscope. This is how we show the cat's eye pattern on track 16 of the AAD and the azimuth bursts on track 34.

Measuring Time Intervals

For short intervals we can read a timer at the beginning of the event and read it again at the end of the event. The difference between the two readings is the time of the event in RTX clock cycles. With an 8 MHz clock, each cycle represents 125 ns (there are 8 cycles per microsecond). The timers roll-over every 65,536 cycles (unless they have been initialized with a smaller value). At 8 MHz this is only about 8 ms.

For longer intervals, we select a convenient value and load the timer. Thereafter it will reload itself with the same value. We set up a timer interrupt routine to count the number of roll-overs between the start and end of the event to be measured.

The Aligner takes two time measurements. The first is the time the drive takes for one complete revolution. This is used to make sure the motor speed is set correctly. This is a long interval of, we hope, 200 ms. This uses the roll-over interrupt mentioned above and shown on screen #3628.

The second is the time between signal peaks of the analog read signal. This is an example of a short interval - on the order of 8 to 500 microseconds. To make this measurement we use the EI3-INTW interrupt routine to store the current timer value in a work area starting at PAD and wait until the peak goes away. To find the time between peaks, we compare the two adjacent timer readings at PAD. See screens #3612 and #3613.

Special Techniques

1. Make It Visible

The RTX with Forth is like a jeep with a winch: there is no excuse for staying stuck in the mud. It is easy to use special utility code to help investigate the hardware or software you're working with. These tools do not need to be present in the final application. For example, a routine might need to complete within a certain number of cycles or you'll miss an event. Here's some code to let you determine exactly how long a segment of code takes to execute.

```
: T   ( -)  0 TC1!  ;            ( start the timer at zero)

: .T  ( -)  TC1@ NEGATE 1-       ( elapsed-time)
            -1 IMR!              ( disable all interrupts)
            EI3 UNMASK           ( enable the serial interrupt)
            U. ." cycles "   ( print the result)
            ABORT  ;             ( halt everything so we can )
                                 ( read it)
```

The word T stands for "time" and is placed at the beginning of the section you want to time. The word .T stands for "print time" and is placed at the end of the section you want to time. It turns off the interrupts (except for the serial line communicating with the host!) and aborts after printing the answer. This lets you put it in the middle of a loop without changing the loop parameters - it will stop after just one pass because of the ABORT. The word 1- compensates for the time spent by T & .T between the two timer readings. It can be found experimentally by running

```
: X1  ( -)   T  .T  ;
```

until you get an answer of zero. This is a quick way to get some information you need *now* without having to count instructions. The point is to toss together a test rather than worrying or guessing. Screen #3637 shows some more examples of timing instructions.

2. Make It Add Up

It is important that every pass through the cat's eye loop take exactly the same number of cycles. The unknown variable was how much time would be stolen by the interrupt handler - it would be different on each pass. A timer can be used, sort of like a memory parity bit, to even the time out. The definition of CE on screen #3630 illustrates this technique, forcing each pass to take 8000 cycles (1 ms). The trick is to set the timer at the beginning of each pass and read it at the end of the pass. From this you know how many cycles to kill to make the entire pass come out exactly right. The constant 7992 was found by using T and .T on the loop with interrupts disabled.

We complete construction and programming of the Floppy Disk Aligner in the next issue of **The Computer Journal** *with the final installment of the series.*

```
scr # 3620
( Vp find peak voltage level of azimuth burst or ce sample)

: Vp ( - dac)
  200 SR!   ( set up the delay value used by ADC)
  ADC  ;    ( measure the voltage)
```

Vp ("vee pee")
Measures the peak voltage over a short interval. This is used for both the azimuth bursts and the cat's eye pattern.

This is our workhorse word to measure wave-form amplitudes.

```
scr # 3621
( AWAIT   Wait for the timer to count down to a specific value)

: AWAIT ( timer1-value -)
  BEGIN     TC1@ OVER U< UNTIL    DROP ;
```

scr # 3921
AWAIT Wait until TIMER1 drops down to the value on the stack. This is used by AZ so the azimuth bursts will be measured at the correct times.

```
scr # 3622
( Measure azimuth burst amplitudes)

: AZ ( - 4th 3rd 2nd 1st) ( amplitudes of the 4 bursts)
  WINDOWS 12 +  ( a)
  3 FOR ( . . . a)  DUP @ SWAP 4 - NEXT DROP ( t4 t3 t2 t1)
   ( above puts the starting time for each window on the stack)
  SYNC 0 TC1!   ( wait for the index pulse and start the timer)
  3 FOR ( <starting times> )
    AWAIT Vp      ( wait for the next window and take a reading)
    R> SWAP 2>R       ( tuck that reading under the loop index)
  NEXT
  2R> 2R> \\ ;      ( retrieve the 4 readings from return stack)
```

scr # 3922
AZ Measure the amplitudes of the four azimuth bursts. The WINDOWS array tells it when to do the measuring. It starts timer1 when the index pulse goes active (SYNC) and AWAITs the correct timer values before taking the readings. Note the \\ at the end of the definition. This forces a separate EXIT instruction. Otherwise, the compiler seems willing to "optimize" the return with the 2R> instruction (which it should not do.)

```
scr # 3623
( SUM   add corresponding numbers in two sets of four)

: SUM           ( a b c d w x y z  -  a+w  b+x  c+y  d+z)
  4 ROLL + >R   ( a b c w x y)
  3 ROLL + >R   ( a b w x)
  ROT +    >R   ( a w)
  + 2R> R> ;
( this will be used by .AZ to average several sets of readings)
```

scr # 3923
SUM Add corresponding numbers in two groups of 4. This is used by the following word to average several azimuth readings.

```
scr # 3624
( .AZ  show azimuth test results)
: .AZ ( -)
  !Vb                                  ( find base voltage)
  CLOCK MARK                          ( find azimuth windows)
  AZ AZ AZ AZ AZ           ( take 5 sets of azimuth readings)
  SUM SUM SUM SUM  ( 4th 3rd 2nd 1st)          ( add them up)
  2  ( ie starting-row for cursor position)
  3 FOR        ( make 4 passes, one for each of the 4 bursts)
    DUP 0 AT          ( position cursor on the next row down)
    SWAP 5 / ( we added up 5 readings so now we take average)
    Vb @ -            ( then subtract the base voltage level)
    FOR ASCII X EMIT NEXT  20 SPACES 1+ ( ie bump cursor row)
         ( print horizontal histogram representing amplitude)
  NEXT DROP  ;
: ST-AZ ( -) 34 SEEK   ['] .AZ  HEART ! ;
```

scr # 3924
Read Azimuth Amplitudes & Show Results

the peak voltages to something (!Vb). Find when to sample (CLOCK & MARK). Take 5 samples and average them. Position the cursor and display a horizontal histogram with 'X's to represent the amplitudes less the base voltage Vb.

ST-AZ Start the azimuth test. Make the azimuth test the active test.

```
scr # 3625
( reduce or divide each sample by a common value )

: LOW-SAMPLE ( - u)  ( find the minimum out of all 200 samples)
  PAD 255 #SAMPLES FOR OVER C@ MIN SWAP 1+ SWAP NEXT
  SWAP DROP ;

: -SAMPLES ( subtrahend -)  SR! PAD  #SAMPLES
  FOR DUP C@ SR@ - OVER C! 1+  NEXT DROP  ;

: /SAMPLES ( divisor -)  SR! PAD #SAMPLES
  FOR DUP C@ SR@ / OVER C!  1+ NEXT DROP ;
```

scr # 3925
LOW-SAMPLE Find the group minimum.

-SAMPLES Subtract a common value from each sample.

/SAMPLES Divide each sample by a common divisor.

These words are used to scale the measurements for display on the terminal. (The samples are stored at PAD by CE.) These words are called by MASSAGE to massage the data.

scr # 3926

Stepper Motor Control with the F68HC11

If One is Good, Two is Better

By Matthew Mercaldo

This segment of our journey will be a deviation from the "Plan"; an exciting one. The original intent of this article was to control a motor with feedback. The feedback was to be some type of encoder or like device which measures the radian travel of the motor, or other feedback device that is attached to the motor. Unfortunately an encoder device could not be found which would fit within the meager budget of this series. Attempts are still being made along these ends.

Instead we will spin two motors, at will and asynchronously, in the most unique way. A framework will be discussed which allows for an extremely efficient use of processor resources and memory. To do this we will use a technique called "Stepped Inference". It is a way to represent state oriented models with a system of rules. This technique will give us the required medium to factor the system states. From these states we can, through techniques of structured design, ascertain a set of rules representative of the system states. In a concise form this set of rules can be embedded into a "State Knowledge Base". If properly designed into the system, this set of rules can be used to run many similar devices asynchronously. The consolidation of device context into some data object or structure (the MCB in the last article) is the key to achieving asynchronous behaviour.

Listing 1

```
HEX
( Article III in a series )
( Stepper Motor Control by Matthew Mercaldo )
( Offset to be added to TCNT - free running timer - for next )
( Timer Output Compare interrupt.                            )
VARIABLE TIMER_OFFSET   -1 TIMER_OFFSET !

( The Motor Control Block structure. )
000 CONSTANT NEXT_MOTOR   ( Pointer to Next MCB )
002 CONSTANT ACTION       ( Pointer to an action to carry out )
004 CONSTANT SENSOR       ( Pointer to a sensor action )
006 CONSTANT >>PORT       ( Pointer to the NMIS 7040 Port )
008 CONSTANT TIMER        ( Pointer to a timer variable )
00A CONSTANT STEP_BIT     ( The active motor bit of the NMIS 7040 )

( More MCB structure elements to the MCB )
00C CONSTANT ACCUMULATOR  ( A timer which is decremented by rateCnt on )
                          ( each interrupt.  When the accumulator's    )
                          ( value reaches zero, a new phase pattern is )
                          ( written to the motor step circuit, and the )
                          ( accumulator is reinstantiated with         )
                          ( stepcnt's value. )
00E CONSTANT RATECNT      ( Determines the velocity of the motor.      )
                          ( This state variable in cooperation with    )
                          ( acclcnt determine the motor's acceleration )
                          ( or deceleration. )
010 CONSTATN ACCLCNT      ( Determines the acceleration, both positive )
                          ( and negative, of the motor. )
012 CONSTANT STEPCNT      ( Determines in cooperation with ratecnt the )
                          ( slowest motor velocity. )
014 CONSTANT +SLOPE       ( Determines the acceleration of the motor. )
016 CONSTANT -SLOPE       ( Determines the deceleration of the motor. )
018 CONSTANT InferenceTable  ( Pointer to the current state Knowledge. )
01A CONSTANT ^InferenceTable ( Pointer to the current   )
                             ( position in the state    )
                             ( Knowledge table.         )
```

Listing continued next page

A Walk Through the Code

The code for startup and the assembler are the same as were used in the last article in this series. Listing 1. is the new code. Some of it will look familiar; some of it will look quite alien. The code in listing one has been grown from the motor code in the last article. Immediately we see that the MCB has changed and grown to allow for the motor speed control context information. Somewhat below, the rules are compiled into the system. Rules are each composed of two parts; the condition routine and the action routine. There are four groups of rules, each group representative of a different motor state. The states represented by rule groups for this system are the initialization state, the acceleration state, the constant velocity state and the done state. Following the rules and their combination into the State Knowledge

See Stepper, page 50

Matthew Mercaldo is employed by a huge firm. With a small group, he develops software tools for field service engineers to do their thing. At 4:30 or 5:00 p.m., when the whistle blows, his thoughts race toward the edge. He dreams of articulated six legged walking beasts, electronic brains that can fend for themselves, and the stuff of "U.S. Robots and Mechanical Men." Someday he dreams of running power out to his garage, and with his wife and a select group of friends, opening his own automoton shop - and thus partially fulfilling his childhood dreams. (Plutonium, Tritium and the like are still not available for public "consumption"; but seeing the moons of Jupiter would be spectacular in one's own starcruiser!)

```
01C CONSTANT CV            ( Determines the length of    )
                           ( motor run.                  )
01E CONSTANT MCB_SIZE      ( Size of this data structure )

000 CONSTANT [CONDITION]   ( Offset into the table of the)
                           ( condition of current rule.  )
002 CONSTANT [ACTION]      ( Offset into the table of the)
                           ( action of the current rule. )

: (CONDITION) CREATE ;     ( Definition of the condition )
                           ( part of a rule.             )
: (ACTION) CREATE ;        ( Definition of the action    )
                           ( part of a rule.             )
: (KB_)  CREATE ;          ( Definition of a state       )
                           ( Knowledge base.             )

8000 CONSTANT >PORT        ( Pointer to NMIS 7040 Port   )

CREATE ANCHOR_MCB MCB_SIZE ALLOT
                ( The Base MCB for a multiple device    )
                ( System.  Each Device has its own MCB. )
                ( Each MCB is linked to another.  At a  )
                ( Regular Heart beat interrupt, this    )
                ( MCB is used to Reference the rest of  )
                ( the list of MCBs.  The list is        )
                ( circular so the anchor MCB's action   )
                ( will also be fired last.  The anchor  )
                ( MCB's action is an RTI.               )
                ( All MCB's actions must fire before    )
                ( the next interrupt fires. )

VARIABLE CURRENT_MCB
                    ( This is the currently active MCB )
                    ( during the MCB interrupt cycle.  )

CREATE FIRST_MCB MCB_SIZE ALLOT
                    ( The MCB we will use for the NMIS )
                    ( 7040 Stepper motor example.      )

VARIABLE >LAST_ACTION   ( Forward reference last_action
)
VARIABLE >SENSOR_ACTION ( Forward reference sensor_action
)

( The Tools will live here... )
( Enable Interrupts; This word enables all Interrupts )
CODE EI  ( - )
ASSEMBLER
   CLI
   NEXT ^ JMP
END-CODE

( Disable Interrupts; This word disables all Interrupts )
CODE DI  ( - )
ASSEMBLER
   SEI
   NEXT ^ JMP
END-CODE

( A stub for future sensor update actions.)
( i.e. encoders, etc.                     )
CREATE SENSOR_ACTION   ( - ; Must be called via JSR )
ASSEMBLER
    RTS

SENSOR_ACTION  >SENSOR_ACTION !

( Steps the motor accessed by X's MCB )
CREATE _STEP ( - ; code to step the motor  )
             (      X must point to MCB    )
             (      uses D                 )
    ( Toggle Step Bit UP )
    STEP_BIT  ,X  A LDA
    A ASL
    >PORT ^  A EOR
    >PORT ^  A STA
    ( Toggle Step Bit Down )
    STEP_BIT  ,X  A LDA
    A ASL
    >PORT ^  A EOR
    >PORT ^  A STA
    RTS
( End _STEP )


( ------------------------- )

VARIABLE >INIT_KB  ( Forward reference to init_kb   )
VARIABLE >ACCL_KB  ( Forward reference to accl_kb   )
VARIABLE >CV_KB    ( Forward reference to cv_kb     )
VARIABLE >DONE_KB  ( Forward reference to done_kb   )

( Rules are made up of two components: a condition  )
( and an action. In the definition of rules, the    )
(following code defines condition and action names  )
(associated with the current state.                 )
( 1.X are the initialization rules.                 )
( 2.X are the acceleration and deceleration rules.  )
( 3.X are the constant velocity rules.              )
( 4.X are the completion rules.                     )
( Rules come in two portions; the even and the odd. )
( The even will be the condition portion of the rule)
( and the odd will be the action portion of the rule)


( ************************************************** )

( RULE 1 of the INIT_KB )
(CONDITION) (1.0)
ASSEMBLER
   -1 # A LDA
   RTS
(ACTION) (1.1)
ASSEMBLER
   STEPCNT     ,X LDD
   ACCUMULATOR ,X STD
   +SLOPE      ,X LDD
   ACCLCNT     ,X STD
   RATECNT     ,X STD

   >ACCL_KB    ^ LDD
   InferenceTable ,X STD
   RTS

( ************************************************** )

( RULE 1 of the ACCL_KB )
(CONDITION) (2.0)
( if accumulator > 0 and )
(    ratecnt > 0         )
ASSEMBLER
   ACCUMULATOR ,X LDD
   PL IF
      RATECNT ,X LDD
      PL IF
         -1 # A LDA
         RTS
      THEN
   THEN
   A CLR
   RTS
(ACTION) (2.1)
( then accumulator = accumulator - ratecnt )
ASSEMBLER
   ACCUMULATOR ,X LDD
   RATECNT     ,X SUBD
   ACCUMULATOR ,X STD
   RTS

( RULE 2 of the ACCL_KB )
(CONDITION) (2.2)
( if ratecnt >= 0       and )
(    ratecnt >= stepcnt and )
(    accumulator < 0        )
ASSEMBLER
   RATECNT ,X LDD
   GE IF
      STEPCNT ,X CPD
      LE IF
         -1 # A LDA
```

```
        RTS
      THEN
   THEN
   A CLR
   RTS
(ACTION) (2.3)
( then ratecnt = ratecnt + acclcnt and )
(      accumulator = stepcnt       and )
(      step the motor                  )
ASSEMBLER
   _STEP ^ JSR
   ACCLCNT      ,X LDD
   RATECNT      ,X ADDD
   RATECNT      ,X STD
   STEPCNT      ,X LDD
   ACCUMULATOR ,X STD
   RTS

( RULE 3 of the ACCL_KB )
(CONDITION) (2.4)
( if ratecnt > stepcnt )
ASSEMBLER
   RATECNT ,X LDD
   STEPCNT ,X CPD
   PL IF
      -1 # A LDA
      RTS
   THEN
   A CLR
   RTS
(ACTION) (2.5)
( then step the motor                  and )
(      set the accumulator to cv count and )
(      set the state Knowledge Base to CV  )
ASSEMBLER
   _STEP ^ JSR
   CV           ,X LDD
   ACCUMULATOR ,X STD

   >CV_KB ^ LDD
   InferenceTable ,X STD
   RTS

( RULE 4 of the ACCL_KB )
(CONDITION) (2.6)
( if  ratecnt <= 0 )
ASSEMBLER
   RATECNT ,X LDD
   LE IF
      -1 # A LDA
      RTS
   THEN
   A CLR
   RTS
(ACTION) (2.7)
( then  set the state Knowledge Base to done  and  )
(       we're done with this instance of motor run )
ASSEMBLER
   >DONE_KB ^ LDD
   InferenceTable ,X STD
   RTS

( ************************************************* )

( RULE 1 of the CV_KB )
(CONDITION) (3.0)
( if  accumulator > 0 )
ASSEMBLER
   ACCUMULATOR ,X LDD
   PL IF
      -1 # A LDA
      RTS
   THEN
   A CLR
   RTS
(ACTION) (3.1)
( then  step the motor )
( note that this is a primitive way of approaching this! )
ASSEMBLER
   _STEP ^ JSR
   ACCUMULATOR ,X LDD
   1 # SUBD
   ACCUMULATOR ,X STD
   RTS

( RULE 2 of the CV_KB )
(CONDITION) (3.2)
( if accumulator <= 0 )
ASSEMBLER
   ACCUMULATOR ,X LDD
   LE IF
      -1 # A LDA
      RTS
   THEN
   A CLR
   RTS
(ACTION) (3.3)
( then  step the motor                          and )
(       set the appropriate state variables     and )
(       set the state Knowledge Base back to accl )
ASSEMBLER
   _STEP ^ JSR

   STEPCNT      ,X LDD
   ACCUMULATOR ,X STD
   -SLOPE       ,X LDD
   ACCLCNT      ,X STD
   STEPCNT      ,X ADDD
   RATECNT      ,X STD

   >ACCL_KB ^ LDD
   InferenceTable ,X STD
   RTS

( ************************************************* )

( RULE 1 of the DONE_KB )
(CONDITION) (4.0)
ASSEMBLER
   -1 # A LDA
   RTS
(ACTION) (4.1)
( The done kb action can be a sort of AST or completion )
( routine.  This would be typical in a multitasker!     )
ASSEMBLER
   -1 # LDD
   ACCUMULATOR ,X STD
   RTS

( ************************************************* )

( The state Knowledge Bases are tables of pointers to )
( rules.  The rules are laid down in the Forth        )
( Dictionary.  When a KB is defined, its reference is )
( instantiated. )

(KB_) INIT_KB
   (1.0) , (1.1) ,
INIT_KB >INIT_KB !

(KB_) ACCL_KB
   (2.0) , (2.1) ,
   (2.2) , (2.3) ,
   (2.4) , (2.5) ,
   (2.6) , (2.7) ,
ACCL_KB >ACCL_KB !

(KB_) CV_KB
   (3.0) , (3.1) ,
   (3.2) , (3.3) ,
CV_KB >CV_KB !

(KB_) DONE_KB
   (4.0) , (4.1) ,
DONE_KB >DONE_KB !

( ------------------------ )
( Engine is the production rule engine.  It is a  )
( recursively propelled little beast.  The main   )
( concideration is that each KB MUST have one rule)
```

```
( that will fire on each Inference cycle.          )
CREATE Engine  ( – ; Stepped Inference Mechanism )
               (      X -> MCB, Y saved on call   )
ASSEMBLER
    ^InferenceTable ,X LDY
    [CONDITION] ,Y LDY
    0 ,Y JSR
    EQ IF
       ( If this condition didn't fire .. Get next )
       ^InferenceTable ,X LDD
       4 # ADDD
       ^InferenceTable ,X STD
       ( and Recurse )
       Engine ^ JSR
    ELSE
       ( This condition fired so do the Action )
       ^InferenceTable ,X LDY
       [ACTION] ,Y LDY
       0 ,Y JSR
    THEN
    RTS
( End Engine )

( doState sets up the registers for the run of this )
( inference cycle. )
CREATE  doState  ( – ; Y has the Current MCB )

    PSHY  PULX            ( Get MCB into X )
    PSHY                  ( Save Y )
    InferenceTable ,X LDD    ( Instantiate ^IT )
    ^InferenceTable ,X STD
    Engine ^ JSR          ( Do inference )
    PULY                  ( Restore Y )

    NEXT_MOTOR ,Y LDY     ( Y -> Next MCB )
    ACTION  ,Y  LDX       ( X -> Action   )
    0 ,X JMP              ( Do It )

( End doState )


( The last MCB's – Anchor MCB's – action )
CREATE LAST_ACTION    ( – ; Returns from processing)
                      ( interrupt )
ASSEMBLER
    PULY
    RTI

LAST_ACTION  >LAST_ACTION !

( Output Compare Timer Interrupt handler routine Specific)
( to Output Compare Timer 1. This Interrupt Routine will )
( process the list of MCB's. The list must be circular!!!)
( ANCHOR -> FIRST -> ... -> LAST -> ANCHOR. The Anchor   )
( MCB's action must end with an RTI. All other MCBs must )
( get the next MCB's action and execute it: This time We )
( will use Register Y to hold the CURRENT_MCB.. )
CREATE -MOTOR_HANDLER ( – ; uses X, D ;; Processes list  )
                      (of MCB's )
ASSEMBLER
   OC1F # A LDA           ( Acknowledge Interrupt. )
   TFLG1 ^ A STA

   TCNT ^ LDD             ( Set Up for next interrupt )
   TIMER_OFFSET ^ ADDD    ( based on Interval from    )
                          ( TIMER_OFFSET. )
   TOC1 ^ STD

   PSHY                   ( Save Forth's Data Stack )
   ANCHOR_MCB # LDY       ( Get Anchor -> Y )
   NEXT_MOTOR ,Y LDY      ( Get First Motor from    )
                          ( Anchor -> Y )

   ACTION ,Y LDX          ( Do First Motor's Action )
   0 ,X JMP

( End of -MOTOR_HANDLER )

: INITIALIZE_MOTOR ( – )
    ( Connect last block to itself, connect last action )
    ANCHOR_MCB  ANCHOR_MCB NEXT_MOTOR + !
    >LAST_ACTION @ ANCHOR_MCB ACTION + !
    ( set the interrupt vector )
    -MOTOR_HANDLER  >TOC1 VECTOR ;

INITIALIZE_MOTOR

CODE ADD_MOTOR_TO_LIST ( mcb – mcb ; uses X, D ;;)
                       ( Adds a new MCB to the   )
                       ( List of MCBs.           )
ASSEMBLER
   ANCHOR_MCB # LDX
   NEXT_MOTOR ,X LDD
   0 ,Y LDX
   NEXT_MOTOR ,X STD

   0 ,Y LDD
   ANCHOR_MCB # LDX
   NEXT_MOTOR ,X STD

   NEXT ^ JMP
END-CODE

CODE FILL_MCB ( mcb action sensor timer port #      )
              ( IT CV stepcnt +slope -slope – mcb   )
              (       ; uses X, D                   )
              (       ; Sets parameters in a new MCB)
ASSEMBLER

    14 ,Y LDX    ( Get MCB from bottom )
    00 ,Y LDD    -SLOPE ,X STD              INY INY
    00 ,Y LDD    +SLOPE ,X STD              INY INY
    00 ,Y LDD    STEPCNT ,X STD             INY INY
    00 ,Y LDD    CV ,X STD                  INY INY
    00 ,Y LDD    InferenceTable ,X STD      INY INY
    01 ,Y A LDA  STEP_BIT ,X A STA          INY INY
    00 ,Y LDD    >>PORT ,X STD              INY INY
    00 ,Y LDD    TIMER ,X STD               INY INY
    00 ,Y LDD    SENSOR ,X STD              INY INY
    00 ,Y LDD    ACTION ,X STD              INY INY
    NEXT ^ JMP
END-CODE


: ADD_MOTOR ( mcb action sensor timer port # – )
   FILL_MCB   ADD_MOTOR_TO_LIST DROP ;

1000 TIMER_OFFSET !


FIRST_MCB           ( MCB )
doState             ( Action Routine                )
>SENSOR_ACTION @    ( Sensor Routine                )
0                   ( Reload Timer count "reserved")
8000                ( Address of Motor Port         )
01                  ( Address in Port of Motor      )
INIT_KB             ( KB for starting up inference  )
100                 ( constant velocity count       )
100                 ( step count constant           )
5                   ( acceleration constant         )
-1                  ( deceleration constant         )
ADD_MOTOR           ( add this motor to list!       )

: LEFT  ( MCB – ; motor left )
   DUP >>PORT + @ C@   ( Get the Port contents on stack )
   OVER STEP_BIT + C@  ( Get Motor control bit on stack )
   OR                  ( OR Port contents and Step bit  )
   SWAP >>PORT + @ C!  ( Put ORd number into Port       )
;

: RIGHT ( MCB – ; motor right )
   DUP >>PORT + @ C@   ( Get Port contents on stack     )
   OVER STEP_BIT + C@  ( Get Motor control bit on stack )
   0FF XOR  AND        ( Clear the relevent bit         )
   SWAP >>PORT + @ C!  ( Put XORd number into Port      )
;

: RUN_MOTOR ( MCB T | F – ; spin it!;)
            ( T is right - F is left )
   DI
```

Modula 2 and the Command Line

Reading the ZCPR Command Line with Modula 2

By David L. Clarke

Introduction

In the past two articles I have shown how Modula 2 can access the Z-System environment and make use of information that it contains such as the TCAP. In this article I shall show how to work with another element in the environment, the Z3 command line. I've added a few extras which may be used optionally. You could say CP/M will never be the same.

Most compilers lately seem to have some method of examining the command line. The FTL compiler that I use at home has a module called Command with a GetParams procedure that supplies the arguments one at a time. It even can expand file names containing wild cards. You might ask why I would want to write a replacement module. For one thing, Command uses the standard CP/M command line at hex address 80, which is shorter that the Z3 command line. It is also not set up for multiple commands on the same line. But for the most part, I think I just wanted to add a few additional features.

The first feature is file redirection. This is not really new to CP/M. I have seen several other programs that use it, especially ones that were written in C. (But why should they have all the fun?) Redirection allows the same program to accept input from the keyboard or from a file. Likewise output can go either to the screen or to a file. Unless otherwise stated, the 'standard input' is from the keyboard and the 'standard output' is the screen. 'Standard input' is redirected to a specific file by putting a '<' character before its name in the command line. A '>' is used to redirect the 'standard output' to a new file. The code within the program does not need to know if the 'standard' input or output is a terminal or a file, a single Read or Write, imported from the InOut module, will work for either.

The second feature is pipelining. This concept was originally introduced with UNIX. It was later copied by MS-DOS. In pipelining, the output of one program is fed into the input of another program. In UNIX, the programs run simultaneously. In MS-DOS, the programs run sequentially. I followed the latter method. In this form, piping is quite similar to having multiple commands in a single command line. The major difference is the way that data is fed from one to the next. Also, multiple commands on a

Listing 1

```
DEFINITION MODULE ZParam;

(*      D. L. Clarke            for TCJ             8 February 1991          *)

(*      This module is designed to provide access to the command arguments  *)
(*    from the Z-environment (rather than the command line).  Some UNIX -   *)
(*    like enhancements have been added.  These include I/O redirection     *)
(*    and a pipeline mechanism.  (Note, 'Exit' must be called to pipeline.) *)

(*      There are several types of arguments.  Any arguments in quotes are  *)
(*    strings and remain as they are.  Arguments preceeded by '<' or '>'    *)
(*    are redirected files (they will be 'redirected' during initialisation *)
(*    and will not be accessible by GetArgument).  Other arguments are      *)
(*    called 'names', they may contain wildcards and will be expanded into  *)
(*    file names if possible.                                               *)

TYPE
    ZResult = (ZDone, ZNotFound, ZTruncated);   (* success/error mnemonics  *)
                                                (*   ZDone = successful     *)
                                                (*   ZNotFound = not a      *)
                                                (*      valid argument      *)
                                                (*   ZTruncated = character *)
                                                (*      array too small for *)
                                                (*      complete argument   *)

PROCEDURE NoOfArguments():      CARDINAL;       (* number of command args   *)

PROCEDURE GetArgument(                          (* get command argument     *)
                      argNr:    CARDINAL;       (* number of desired arg    *)
                                               (*   note - 0 = prog name    *)
                VAR     arg:    ARRAY OF CHAR; (* the returned argument     *)
                VAR result:     ZResult);       (* success / error code     *)

PROCEDURE Exit(                                 (* exit pipeline program    *)
                      code:     INTEGER);       (* exit error code          *)
                                                (*   note - 0 = no error    *)

END ZParam.
```

David Clarke was originally an Electrical Engineer at Pratt & Whitney Aircraft until he discovered that it was more fun to program the data acquisition systems that he developed. He therefore became a systems programmer.

Dave is also an Adjunct Professor at the Hartford Graduate Center in Hartford CT., where he has taught courses in Systems Programming, Software Engineering, and Real Time Programming. Dave can be reached at the Graduate Center where his electronic mail address (Internet) is davec@mstr.hgc.edu. His home address for regular mail is P.O. Box 328, Tolland, CT. 06084.

Listing 2

```
IMPLEMENTATION MODULE ZParam;

(*      D. L. Clarke            for TCJ         8 February 1991         *)

(*      This module is loosely based on the 'Command' module supplied   *)
(*    originally with the FTL compiler.  The concepts have been         *)
(*    expanded to include the Z-environment and some UNIX like ideas.   *)

FROM  Files      IMPORT  FILE, FileName, Lookup, Create, Rename, Close;
FROM  GetFiles   IMPORT  GetNames;
FROM  InOut      IMPORT  SwitchInStream, SwitchOutStream;
FROM  SeqBuffer  IMPORT  Sequence, Access, Deaccess, GetSeq, PutSeq, Delete,
                         Highest, Lowest;
FROM  SmallIO    IMPORT  WriteInt;
FROM  Streams    IMPORT  STREAM, Direction, Connect, Disconnect,
                         EOS, ReadChar, WriteChar;
FROM  Strings    IMPORT  Length, CopyStr, Insert;
FROM  Terminal   IMPORT  WriteString, WriteLn;
FROM  SYSTEM     IMPORT  ADR;
FROM  z34m2      IMPORT  CL, Z3PTR, Z3ENV, GetEnv;

    TYPE ParClass = (Pipe, CommandEnd, String, RedirectIn, RedirectOut,
Name);

    VAR Param:          RECORD
                            Class:  ParClass;
                            Chars:  CL
                        END;
        buff:           Sequence;
        ProgramName:    ARRAY [0..7]   OF CHAR;
        Names:          ARRAY [0..26]  OF FileName;
        ArgList:        ARRAY [0..100] OF RECORD
                            index:  INTEGER;
                            name:   FileName
                        END;
        env:            Z3PTR;
        new_CL:         CL;
        F_Name:         FileName
        InF, OutF:      FILE;
        InS, OutS:      STREAM;
        ArgCount:       CARDINAL;
        i, j, k:        CARDINAL;
        NameCount, err: INTEGER;
        InRedirected:   BOOLEAN;
        OutRedirected:  BOOLEAN;
        Pipeline:       BOOLEAN;
        done:           BOOLEAN;

    PROCEDURE NoOfArguments(): CARDINAL;
    BEGIN
        RETURN ArgCount
    END  NoOfArguments;

    PROCEDURE GetArgument(argNr: CARDINAL;  VAR arg: ARRAY OF CHAR;
                      VAR result: ZResult);
    BEGIN
        IF (argNr < 0) OR (argNr > ArgCount) THEN
            result := ZNotFound
        ELSE
            result := ZDone;
            IF ArgList[argNr].index = -1 THEN
                j := CopyStr(ArgList[argNr].name, arg);
                k := Length(ArgList[argNr].name)
            ELSIF GetSeq(Param, ArgList[argNr].index, buff) THEN
                j := CopyStr(Param.Chars, arg);
                k := Length(Param.Chars)
            ELSE
                result := ZNotFound
            END;
            IF (result = ZDone) AND (j < k) THEN
                result := ZTruncated
            END
        END
    END GetArgument;

    PROCEDURE Exit(code: INTEGER);
    BEGIN
        IF code = 0 THEN
```

line are separated by a ';' character while programs that use pipelining are separated by the '|' character. This description may sound rather simple, but don't let it fool you. Piping adds a lot more functionality than you would first think. Hopefully I'll be able to demonstrate its utility within the course of this (and other) articles.

The ZParam Module

The definition module for the ZParam module is shown in *Listing 1*. The primary procedure, of course, is GetArgument. It is requested to return a particular argument from the command line, however, it may have some problems. One one thing, the requested argument may not exist. For another, the ARRAY OF CHAR supplied to receive the argument may be too small for all of the characters in the argument. Therefore the GetArgument procedure returns a result of type ZResult. The three possible return values are enumerated at the beginning of the definition module.

The NoOfArguments procedure is supplied to tell how many arguments exist on the command line (this could possibly be used to prevent a ZNotFound result).

The final procedure in the definition module need only be used by programs that pipeline. The Exit procedure controls the passing of this program's output to the next program's input. However, if a non-zero code is passed to Exit, the procedure will break the pipeline and type out an error message on the terminal.

Listing 2 shows the ZParam implementation module. All of the real work of the module is done during the initialisation. Since this module will be using the Z-System environment, the first thing to do is access the environment by calling GetEnv. If the environment does not exist we type an error and exit, otherwise we continue. The individual parameters are parsed from the command by GetParams and stored in a Seqbuffer named 'buff'. The contents of 'buff' are searched looking for redirected input and output by calling CheckInput and CheckOutput. Finally, all ambiguous file references are expanded and the results are transferred to the ArgList array. This is done by the ExpandNames procedure.

GetParams parses the command line by scanning it character by character. Spaces are skipped until a non-space is

found. This character might be one of a set of key characters. They are ';', '|', '<', '>', '"', and "'". A ';' indicates the end of the current command, anything after it belongs to some other command and is therefore ignored. Likewise a '|' indicates the end of the current command, but in this case the command is part of a pipeline. In each case, the type of the terminating condition (i.e., ParClass) is saved in the 'buff' SeqBuffer. The '<' and '>' characters introduce redirection filenames. The next name is scanned and saved with the appropiate ParClass code. Any text contained in quotes is considered a 'String' and is saved as such. All other sequences of characters are considered to be a 'Name' (i.e., it could be a filename). Commas are just plain skipped. When GetParams has completed, all parameters have been placed in the 'buff' Seqbuffer.

CheckInput and CheckOutput scan through the parameters in 'buff'. When a redirected name is encountered, the file is opened and connected to the appropiate input / output stream. After this, any calls made to InOut module procedures will be directed to the redirected file. Once the connection has been made, the entry is deleted from 'buff', it must not be accessed by GetArgument. CheckOutput also scans for a ParClass of 'Pipe'. If it is detected, an output file named 'STDOUT.$$$' is opened and connected to the output stream. In addition, the next command has a "<STDIN.$$$" string inserted into it. This is how the information shall be passed through the pipeline. The FTL Streams module is used to advantage in this procedure.

The ExpandNames procedure uses the FTL GetFiles module to expand ambiguous names into all the matching filenames. Only parameters of ParClass 'Name' are expanded, 'Strings' are not. The results are transferred sequentially to the ArgList array. This array is where GetArgument will find them. ExpandNames also keeps track of the argument count as it moves the parameters into ArgList. This count is used by the NoOfArguments procedure.

The paragraph directly above explains the operation of NoOfArguments and GetArgument. The definition module also mentioned Exit. If the code passed to Exit is non-zero, it will cause the program to halt with an error message. Otherwise, the 'standard output' file is closed and renamed to

```
        IF InRedirected  THEN Disconnect(InS,  TRUE) END;
        IF OutRedirected THEN
            Disconnect(OutS, TRUE);
            IF Pipeline THEN
                env^.z3cl^.nxt := ADR(env^.z3cl^.str);
                k := CopyStr(new_CL, env^.z3cl^.nxt^);
                env^.z3cl^.len := CHR(k);
                Lookup(OutF, "STDOUT.$$$", err);
                IF err >= 0 THEN
                    Rename(OutF, "STDIN.$$$", "STDOUT.$$$", err)
                END;
                IF err < 0 THEN
                    WriteString(ProgramName);
                    WriteString(": Cannot rename STDOUT.$$$");
                    WriteLn
                ELSE
                    Close(OutF)
                END
            END
        END
    ELSE
        WriteString(ProgramName);
        WriteString(": error exit = ");
        WriteInt(code, 1);    WriteLn
    END;
    HALT
END Exit;

PROCEDURE Save(size: CARDINAL; class: ParClass);
    VAR  j: INTEGER;    k: CARDINAL;
BEGIN
    Param.Class := class;
    FOR k := 0 TO size - 1 DO
        Param.Chars[k] := env^.z3cl^.str[i + k]
    END;
    Param.Chars[size] := 0c;
    j := Highest(buff);    IF j < 0 THEN j := -1 END;
    done := PutSeq(Param, j + 1, buff)
END Save;

PROCEDURE Scan;
BEGIN
    j := i + 1;
    WHILE (j < ORD(env^.z3cls))     AND
          (env^.z3cl^.str[j] # 0c)  AND
          (env^.z3cl^.str[j] # ' ') AND
          (env^.z3cl^.str[j] # '|') AND
          (env^.z3cl^.str[j] # ',') AND
          (env^.z3cl^.str[j] # ';')  DO
        INC(j)
    END
END  Scan;

PROCEDURE GetParams;
BEGIN
    i := 0;
    LOOP
        WHILE (i < ORD(env^.z3cls)) AND (env^.z3cl^.str[i] <= ' ') DO
            IF env^.z3cl^.str[i] = 0c THEN EXIT ELSE INC(i) END
        END;
        IF  i >= ORD(env^.z3cls)  THEN EXIT END;
        CASE env^.z3cl^.str[i] OF
          '|':      (* pipeline to next program *)
                    INC(i);
                    Save(ORD(env^.z3cls)-i, Pipe);
                    j := ORD(env^.z3cls)
        | ';':      (* end of command in multi command *)
                    INC(i);
                    Save(ORD(env^.z3cls)-i, CommandEnd);
                    j := ORD(env^.z3cls)
        | '"', "'": (* quoted string *)
                    j := i + 1;
                    WHILE (j < ORD(env^.z3cls))     AND
                          (env^.z3cl^.str[j] # 0c)  AND
                          (env^.z3cl^.str[j] # env^.z3cl^.str[i]) DO
                        INC(j)
                    END;
                    INC(i);     Save(j-i, String);
                    IF (env^.z3cl^.str[j] = env^.z3cl^.str[i-1]) THEN
                        INC(j)
```

```
                END
        | '<', '>': (* redirected input / output *)
                done := env^.z3cl^.str[i] = '<';
                INC(i);
                WHILE (i < ORD(env^.z3cls))      AND
                      (env^.z3cl^.str[i] > 0c)  AND
                      (env^.z3cl^.str[i] <= ' ') DO
                    INC(i)
                END;
                Scan;
                IF done THEN
                    Save(j-i, RedirectIn)
                ELSE
                    Save(j-i, RedirectOut)
                END
        ELSE    (* run of the mill parameter *)
                Scan;
                Save(j-i, Name)
        END;  (* case *)
        i := j;
        IF env^.z3cl^.str[i] = 0c THEN EXIT END;
        IF (i < ORD(env^.z3cls)) AND (env^.z3cl^.str[i] = ',') THEN
            INC(i)
        END
    END  (* loop *)
END GetParams;

PROCEDURE CheckInput;
BEGIN
    i := 0;    InRedirected := FALSE;
    LOOP  (* search for input redirection *)
        IF NOT GetSeq(Param, i, buff) THEN EXIT END;
        IF i = 0 THEN j := CopyStr(Param.Chars, ProgramName) END;
        IF Param.Class = RedirectIn THEN
            IF InRedirected THEN
                WriteString(ProgramName);
                WriteString(": Multiple input redirection");
                WriteLn;    Exit(-1)
            END;
            InRedirected := TRUE;
            j := CopyStr(Param.Chars, F_Name);
            Lookup(InF, F_Name, err);
            IF err < 0 THEN
                WriteString(ProgramName);
                WriteString(": Cannot open ");   WriteString(F_Name);
                WriteLn;    Exit(err)
            END;
            Connect(InS, InF, input);
            SwitchInStream(InS);
            done := Delete(i, buff)
        END;
        INC(i)
    END  (* loop *)
END  CheckInput;

PROCEDURE CheckOutput;
BEGIN
    i := 0;     OutRedirected := FALSE;    Pipeline := FALSE;
    LOOP    (* search for output redirection *)
        IF GetSeq(Param, i, buff) AND
           ((Param.Class = RedirectOut) OR (Param.Class = Pipe)) THEN
            IF OutRedirected THEN
                WriteString(ProgramName);
                WriteString(": Multiple output redirection");
                WriteLn;    Exit(-1)
            END;
            OutRedirected := TRUE;
            IF Param.Class = RedirectOut THEN
                j := CopyStr(Param.Chars, F_Name)
            ELSE
                Pipeline := TRUE;
                j := CopyStr("STDOUT.$$$", F_Name)
            END;
            Create(OutF, F_Name, err);
            IF err < 0 THEN
                WriteString(ProgramName);
                WriteString(": Cannot open "); WriteString(F_Name);
                WriteLn;    Exit(err)
            END;
            Connect(OutS, OutF, output);
```

'STDIN.$$$'. As described above, CheckOutput has already insured that the next command will redirect its input to this file. Thus, the piping function is accomplished fairly easily.

Compilation hint—I have found that this module needs all the memory it can get to compile. This means I have to compile it under bare CP/M; ZCPR must be unloaded.

Modifications to the Strings Module

One of the things that I like best about the FTL compiler is that it supplies the sources to its library modules. This allows you to modify them to suit your needs. One place where I've made some changes is in the Strings module. I chose to replace the supplied Assign, StoS, and Copy procedures by a single procedure called CopyStr. I also added a procedure that compares two strings alphabetically which is called CompareStr. The definition module, as I use it, is shown in *Listing 3*. The implementation code for the two modified procedures is shown in *Listing 4;* note, this is not a complete listing of the module, the other procedures are basically unchanged. When looking at CompareStr, you may notice that it uses the CaseSensitive boolean to determine how alphabetic characters are compared. This variable appears in the definition module and is set to TRUE during the module initialisation. The Pos procedure was also modified to be case sensitive.

Two Sample Programs

Listing 5 shows my version of the 'echo' program. As with the Z-System program it uses '%>' to start echoing in lower case (and '%<' to return to upper case). Therefore, a command of

```
echo h%>ello world
```

will output "Hello world". On the other hand, my version will expand ambiguous filenames (much like UNIX). Thus a command of

```
echo *.*
```

will give you a listing of your current directory, however it will attempt to place it all on a single line (which could cause problems on your terminal). If you don't want wild cards expanded, you should enclose that parameter in double or single quotes — this disables the expansion.

Listing 6 shows my version of the

popular 'cat' program. It concatenates several input files into a single output file (i.e., 'standard output', which may either be the screen or be redirected to a file). If no files are listed as inputs on the command line, the 'standard input' is used, which could be the keyboard or an input from a pipeline. So if you're annoyed that 'echo' outputs the command line arguments too fast or if you like the reassuring sound of disk drives clicking, try a command line like

```
echo "hello world" | cat | cat
| cat >greeting.txt
```

and then type out the contents of greeting.txt

It is also possible to combine pipeline programs with standard (non-pipeline) programs on the same command line. For instance, the 'type' command does not pipeline, and the new 'echo' can either pipeline or write to the 'standard output'. The following command line shows how information may be fed from one program to another.

```
echo "hello world"
>greeting.txt ; type greeting.txt
```

This command line forces 'echo' to produce an output file that can be read by 'type'. The ';' character is used to separate the commands in this form.

Compilation hint—don't forget to M2instal echo and cat after linking them. Otherwise they will not be able to access the Z-System environment. Refer to my article in *TCJ* # 47 for more information on M2instal.

Conclusion

In this article I have introduced a new module that gives us access to the command line and provides several useful options. I have built this upon a foundation of earlier modules (SeqBuffer from the first article, z34m2 from the second article, and many of the modules supplied with the compiler itself).

A glance at the two sample program listings (cat and echo) will show that, for all their capability, the main program code is fairly small. Hopefully this demonstrates how straightforward programming is in Modula 2 when we have a foundation of good modules.

For more information on redirection and piping I would suggest reading "Software Tools in Pascal" by Kernighan and Plauger, Addison-Wesley, Reading, MA (1981). The text has versions of echo and concat which are worth comparing with the listings in this article. Another interesting point about the book is to see how much effort the authors had to go to just to force Pascal into doing things that Modula 2 can do quite easily. Perhaps it's time for a book titled "Software Tools in Modula 2".●

```
                SwitchOutStream(OutS);
                IF Pipeline THEN
                    k := 0;
                    WHILE Param.Chars[k] <= ' ' DO  INC(k)  END;
                    WHILE Param.Chars[k] >  ' ' DO  INC(k)  END;
                    Insert(" <STDIN.$$$ ", Param.Chars, k);
                    k := CopyStr(Param.Chars, new_CL)
                END;
                done := Delete(i, buff)
            END;
            INC(i);
            IF i > CARDINAL(Highest(buff)) THEN EXIT END
        END  (* loop *)
    END  CheckOutput;

    PROCEDURE ExpandNames;
    BEGIN
        i := 0;                 (* ArgList index *)
        j := 0;                 (* Param index   *)
        LOOP
            IF GetSeq(Param, j, buff) THEN
                IF Param.Class = Name THEN
                    GetNames(Param.Chars, Names, NameCount);
                    k := 0;     (* Names index *)
                    LOOP
                        ArgList[i].index := -1;
                        ArgList[i].name  := Names[k];
                        INC(i);          INC(k);
                        IF (i > 100) OR
                           (k >= CARDINAL(NameCount)) THEN EXIT END
                    END
                ELSIF Param.Class = CommandEnd THEN
                    EXIT
                ELSE
                    ArgList[i].index := j;
                    INC(i)
                END
            END;
            INC(j);
            IF (i > 100) OR (j > CARDINAL(Highest(buff))) THEN EXIT END
        END;  (* loop *)
        ArgCount := i - 1
    END ExpandNames;

BEGIN
    env := GetEnv();
    IF CARDINAL(env) = 0 THEN
        WriteString("<ZParam> No Z-Sys Env");    WriteLn;
        HALT
    ELSE
        buff := Access("Parameter buffer");
        GetParams;
        CheckInput;      CheckOutput;
        ExpandNames;
    END
END ZParam.
```

Home Control. from page 26

wiring in my house. Given my interests in electrical experiments, I am lucky to live in a house with an unfinished basement. The electrical wiring (and the plumbing) are conveniently exposed in the ceiling. All the wall outlets were fed from the basement, and it was easy to install GE relays to control those outlets I wished to switch. At the time, our house also had an only partially finished second floor. The few circuits (overhead lights) that I could not tap into from the basement I could get to from the attic.

Shortly after I built the home controller, I renovated the second floor. Actually, I gutted it and rebuilt it from scratch, a project that took a whole year of full-time evening and weekend work. It cured me of any desire ever to do that again, but it did give me a golden opportunity to wire things

See Home Control. page 22

Listing 3

```
DEFINITION MODULE Strings;

(*      D. L. Clarke        for TCJ          (modified) 23 Sept 1990      *)

(*      This is a modification of the Strings module supplied with the  *)
(*   FTL compiler.  Modifications were made to be compatible with some  *)
(*   other versions as well as a proposed 'standard'.                   *)

TYPE    String = ARRAY [0..80] OF CHAR; (*  here for compatibilty only  *)
                                        (*  upper limit need not be 80  *)
                                        (* Note, strings are terminated *)
                                        (* by a zero byte or by length. *)

VAR    CaseSensitive:   BOOLEAN;        (* set to FALSE if you do not   *)
                                        (* want case sensitive compares *)
                                        (*   pre-set to TRUE            *)

PROCEDURE Length(                       (* find length of string        *)
                  s1:   ARRAY OF CHAR)  (* the string                   *)
                   :    CARDINAL;      (* the number of characters     *)

PROCEDURE Pos(                          (* find position of substring   *)
                Match,                  (* the substring to be found    *)
               Search:   ARRAY OF CHAR; (* the string tho be searched   *)
                Start:   CARDINAL)      (* number of characters to skip *)
                   :     CARDINAL;      (* the position in the string   *)
                                        (* =HIGH(Search)+1 if not found *)

PROCEDURE Insert(                       (* insert substring into string *)
              SubStr:   ARRAY OF CHAR;  (* the substring                *)
          VAR    Str:   ARRAY OF CHAR;  (* the receiving string         *)
               Start:   CARDINAL);      (* number of characters to skip *)

PROCEDURE Delete(                       (* delete substring from string *)
          VAR    Str:   ARRAY OF CHAR;  (* the original string          *)
               Start:   CARDINAL;       (* number of characters to skip *)
                 Len:   CARDINAL);      (* length of substring          *)

PROCEDURE CopyStr(                      (* copy string                  *)
              Source:   ARRAY OF CHAR;  (* the source string            *)
          VAR   Dest:   ARRAY OF CHAR)  (* the destination string       *)
                   :    CARDINAL;      (* number of characters copied  *)

PROCEDURE CompareStr                    (* alphabetic string comparison *)
          ( st1, st2: ARRAY OF CHAR)    (* the strings being compared   *)
                    : INTEGER;          (* the resulting comparison     *)
                                        (* < 0 means st1 < st2          *)
                                        (* = 0 means st1 = st2          *)
                                        (* > 0 means st1 > st2          *)

PROCEDURE Concat(                       (* concatenate strings          *)
              s1, s2:   ARRAY OF CHAR;  (* the source strings           *)
          VAR     s3:   ARRAY OF CHAR); (* the destination string       *)
                                        (* truncated if both won't fit  *)

END Strings.
```

Home Control. from page 21

the way I wanted to. I installed numerous switch boxes and ran large bundles of wires from each one back to central control panels. There, as I described earlier, I can easily rewire any of the switches to perform any functions I want. I also fed a huge bundle of low-voltage cables down into the basement. Some go to the computer and some to switches on the first floor. Some are spares for future use.

That completes the material I wanted to cover for this issue. I'm not sure right now what part of the project I will treat next time, but I think it will probably include some of the software.

You may have noticed that I have not included detailed schematics of the circuits I have described. There are two reasons for this. First, I do not have the tools to produce publication-quality circuit diagrams with a reasonable effort. Second, these designs are quite old at this point, and, if I were starting on this project today, in most cases I would not use the exact same circuits. Therefore, I am more interested in presenting ideas and concepts and leaving implementation details to the reader. If anyone is interested in pursuing this type of project, I would be delighted to provide more delighted to provide more detailed information and discussion. Just contact me in one of the usual ways indicated on the side-bar.●

8031 µController Modules

NEW!!!

Control–R II

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8 K of EPROM
√ Socket for 8 Kbytes of Static RAM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O plus access to address, data and control signals on standard headers.
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 3.50" x 4.5" size
√ Assembled & Tested, not a kit

$64.95 each

Control–R I

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8K EPROM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 2.75" x 4.00" size
√ Assembled & Tested, not a kit

$39.95 each

Options:
- MAX232 I.C. ($6.95ea.)
- 6264 8K SRAM ($10.00ea.)

Development Software:
- PseudoSam 51 Software ($50.00) Level II MSDOS cross–assembler. Assemble 8031 code with a PC.
- PseudoMax 51 Software ($100.00) MSDOS cross–simulator. Test and debug 8031 code on your PC!

Ordering Information:
Check or Money Orders accepted. All orders add $3.00 S&H in Continental US or $6.00 for Alaska, Hawaii and Canada. Illinois residents must add 6.25% tax.

Cottage Resources Corporation
Suite 3–672, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

A Home Heating & Lighting Controller, Part 2

The Electrical Interface

By Jay Sage

In the introductory column last time on the embedded controller that runs the electrical and heating systems in my house, I described the history behind its development and the control strategy it applies to managing the heating system. This time I am going to talk about the electrical interface, that is, how the computer, which runs on low-voltage DC, is able to operate the high-voltage AC electrical circuits in the house.

The GE Low-Voltage Wiring System

The basis of the interface is the General Electric low-voltage wiring system. It was designed primarily for industrial installations, but somehow my group leader at Raytheon had found out about it and used it when he built his own house. Since then I have seen it several other times. In fact, the lighting system in the main auditorium at Lincoln Laboratory uses it.

The basic idea is as follows. Each circuit, no matter from how many places it is to be controlled, has a single, latching relay installed to switch the AC current. These relays have two windings that operate at 24 volts, AC or DC.When one of the windings is energized, a shorting bar is pulled back from a pair of electrical contacts, opening the 110-volt circuit; when the other winding is activated, the shorting bar is pushed into the contacts, closing the circuit.

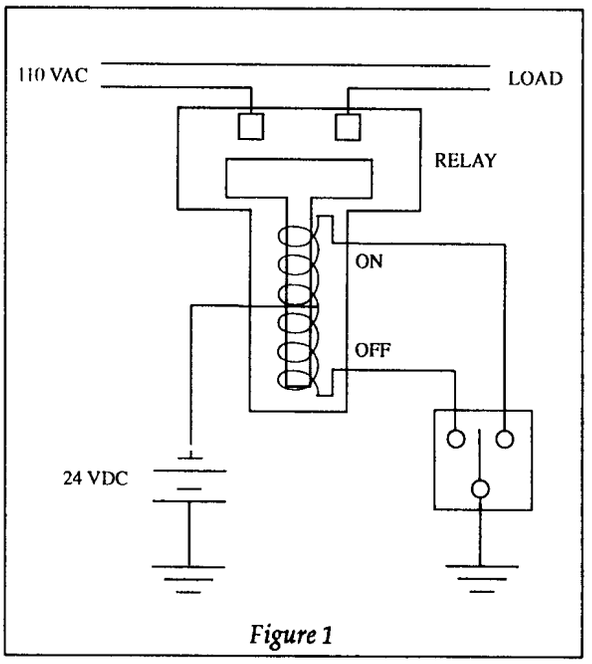


Figure 1

These relays are designed to mount through the cutout in a standard electrical wiring box. The 110-volt AC connections are made inside the box, while the low-voltage connections are made outside. This maintains isolation between the two systems and keeps the low-voltage system safe against shock hazard.

The single-pole single-throw toggle switches normally used in the wall boxes to control the AC are replaced by small, momentary-contact, single-pole double-throw (SPDT), center-off low-voltage switches. The basic circuit is shown in *Figure 1*. For reasons that will become clear later, we operate the switches from a DC source.

At rest, the switches are in a neutral, center position with the pole connected to neither side. To turn the AC circuit on, one presses the top of the switch. This completes the circuit to the ON winding of the relay. The switch need be closed for only a small fraction of a second, after which the relay is latched into the ON state. To turn the circuit off, one taps the bottom of the switch, completing the circuit to the OFF winding of the relay and opening the circuit. The simplest, least expensive low-voltage switches are so small that three of them can fit in the standard electrical box used for one AC switch.

Advantages of the GE Low-Voltage System

The GE low-voltage wiring system has many advantages. For example, there are very strict codes that apply to the treatment of 110-volt AC wiring. For obvious reasons, the wiring has to be carefully protected—mechanically, electrically, and thermally. All connections must be made inside boxes that meet the standards of the electrical code, and wire runs have to be enclosed, either inside walls or inside conduits, or both. Because of the high electrical energy available in 110-volt circuits, overload breakers are required to interrupt the current flow if it reaches a level that could cause the wires to reach an unsafe temperature and ignite a fire. In many, if not all, jurisdictions, 110-volt wiring must be installed by a licensed electrician; at the least, a permit is required, and the work must be inspected. After all, if it is done improperly, it can pose the dual threats of electrocution or fire.

The low-voltage wiring is intrinsically safe. The voltage is too low to cause electrocution, and the transformer that provides the power has a high enough internal impedance that even if the wiring is shorted it cannot generate enough heat to pose a fire hazard. Consequently, there are, as far as I know, few or no code requirements on how this wiring is installed. It's basically like wiring up a doorbell.

In principle, the wiring materials are also cheaper. For 20-ampere service, the AC lines must use 12-gauge wires—including a ground as well as a neutral lead—with heavy insulation. The low-voltage wiring needs only 16- or 18-gauge wire and lighter insulation. No grounding is

necessary.

The switches are also intrinsically much cheaper, since they switch relatively low currents at low voltages. In practice, however, I have found that most of this advantage is hard to realize. Standard AC wiring components, because of the extremely high production volume and competition among suppliers, are available at very low prices. For large-scale industrial installations, there probably is a cost saving, both in material costs and in labor for installation. For home installations, this is not a significant factor.

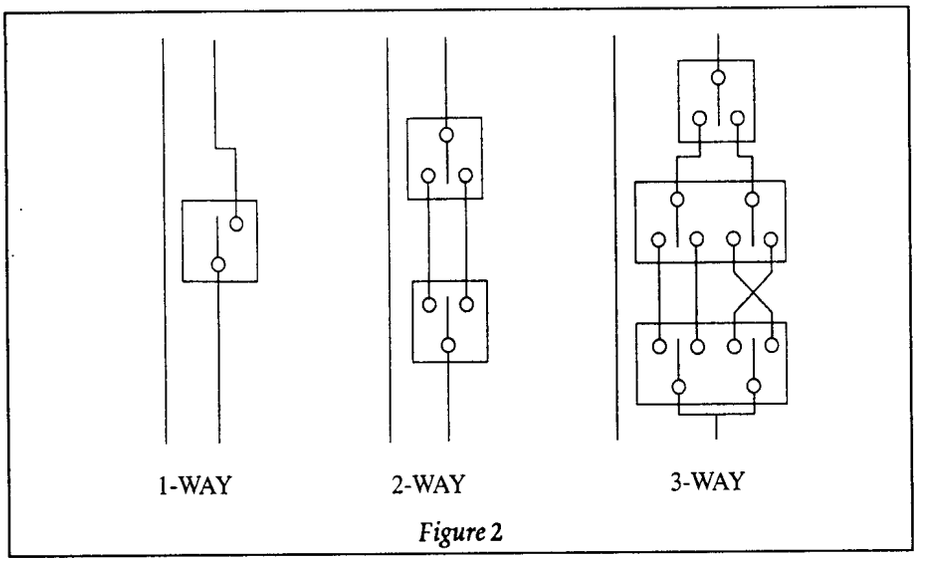


Figure 2

One of the most important advantages of the GE system to me was the ease with which it allows multiple controls in a circuit. Consider the problem of wiring a circuit so that it can be switched from two or three different places. *Figure 2* shows the required switches and wiring for one-, two-, and three-switch circuits. You can see how rapidly they grow in complexity.

With only one switch, a single-pole single-throw (SPST) switch suffices, along with a normal three-wire cable (hot, neutral return, and ground). The two-way circuit requires single-pole double-throw (SPDT) switches, and the cable between the two switches has to have and extra conductor. By the time we get to the three-way circuit, we have *double*-pole double-throw switches (DPDT) and up to *four* extra wires in the cable.

The same circuit using the GE wiring is illustrated in *Figure 3*. Note the extreme simplicity. Although I have shown only three switches, in fact there can be *any number of switches*. The circuit does not increase in complexity no matter how many switches are used. This is a bus architecture—if you want another switch, you just hang it off the bus!

And this just begins to show the flexibility of the GE system. Look at *Figure 4!* There we show two AC circuits. Switch A controls the circuit on the left; switch C controls the circuit on the right. What about switch B? It controls *both* circuits. The diodes are needed to isolate the two relay control circuits (and that's why we use a DC low-voltage source). If you study the circuit, you will see that when switch A is closed, it cannot draw current from a winding of the right-hand relay because there will be a reversed diode in the path. With conventional AC switches, this functionality can be achieved only by ganging up electrically independent switches, a bulky and expensive proposition. With the GE system, the only extra components are a few tiny, dirt cheap switching diodes.

The concept can be extended to a complete diode crossbar connecting a group of switches 1...M to a group of relay windings 1...N, as shown in *Figure 5*. By inserting diodes into the crossbar, any switch can be made to control any combination of relay windings, completely independently of the function of any other switch. When a given switch grounds one of its leads, all relay windings connected to that lead by a diode will be pulled to ground, energizing the associated relay. The directionality of the diodes prevents one pulled-down relay winding from pulling down another switch wire and thence other relay windings.

Note that the ON and the OFF functions of a single switch may be programmed independently! For example, one of the switches in our bedroom is set up to turn off all three lights in the living room. This is handy when we go to bed. The ON position of the same switch, however, turns on only one light. Separate switches in or near the living room control the individual lamps. In fact, there is nothing to say that an OFF position on a switch has to turn a circuit off; it can just as well turn it on. One might have a switch whose ON position turns one light on and another one off. The OFF position, meanwhile, might turn the second light on and the first one

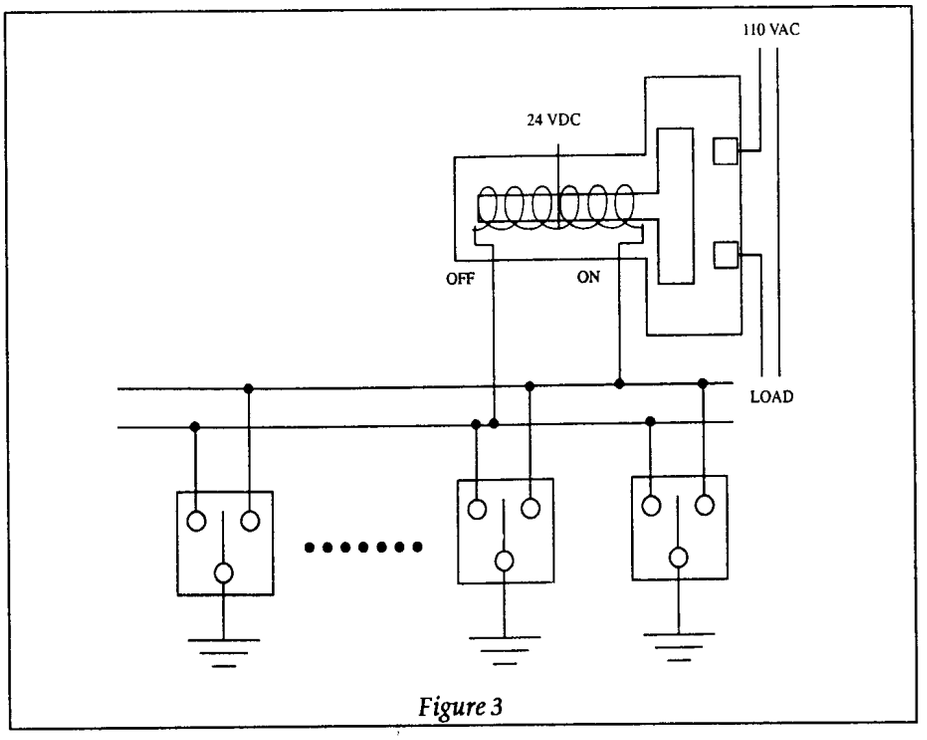


Figure 3

off. Basically, you can do absolutely whatever you want.

Interfacing to the Computer

The next important thing to realize is that switches are not the only things that can activate the relay windings. All the switches are doing is pulling a circuit down to ground, and that is something that a *transistor* can do just as well.

It would be very hard to use a transistor to control AC

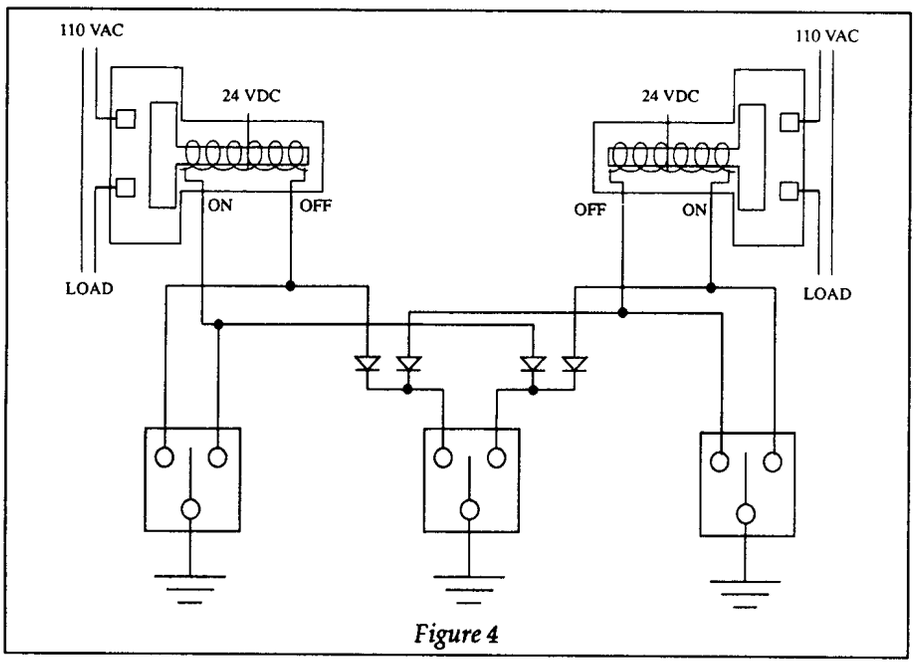


Figure 4

circuits directly. First of all, transistors cannot easily control such high voltages and currents. Secondly, transistors are intrinsically unipolar; they like to have current flowing in only one direction. For example, with an NPN transistor, current injected into the base controls a current flowing into the collector toward the emitter. The magnitude of the current can be controlled but not its direction.

Although the relays latch at modest currents (fractions of an ampere), the normal parallel-port drivers on a computer are not, by themselves, powerful enough to operate the relays. However, there are very nice, inexpensive, compact integrated circuits that provide the buffering between the milliampere drive capability of parallel-port chips and ampere-range currents of peripheral circuits.

I used ULN-2814 ICs made by the Sprague Electric Company. These are 18-pin DIPs with eight individually controlled circuits, each capable of sinking 600 ma of current and sustaining up to 50 volts in the OFF state. I can't find the data sheet for this specific part, but similar parts for which I do have data sheets require input currents of less than 1 ma.

More Details on the Interface

I would now like to cover some details of the interface circuit, which is actually a little more complicated than what I suggested above.

I designed the controller to handle 16 circuits, which means 32 relay windings. This would take four 8-bit parallel ports to handle all at once, but I decided that the controller would pulse no more than one relay at a time. This simplified the control software, since circuits can then be treated sequentially, one at a time. In addition, it completely eliminates the possibility of exceeding the power dissipation limits of the driver ICs, since only one of the eight drivers could be dissipating power at any one time.

If only one driver is to be active at one time, I can get by with an addressing circuit. Five bits are enough to select one driver out of 32. Since most of the time no circuits are activated, we need one more line to serve as an 'enable' control. That makes six lines in all; one parallel port from the microcomputer can handle it.

It seemed a bit risky to me to have the relay windings connected directly to the computer. For one thing, inductive circuits, such as relay coils, tend to produce nasty transients. Although I included in the circuit some protective diodes to clip any voltages that tried to get outside the range from ground to 24 volts, I felt that the computer was too expensive to take any chances with. The low-voltage wiring involved hundreds or even thousands of feet of wire, and there was always a chance for an error that might put a very high voltage on that wiring.

The answer was to optically isolate the computer from the relay-driver circuitry. For those of you who are not familiar with these components, they have a photoemitter (light-emitting diode or LED) controlled by the input side of the IC and a photodiode or phototransistor detector on the output side of the IC. The two parts are separated by a high-dielectric strength insulator that is optically transparent.

I chose the 5082-4351 optically coupled isolators from Hewlett Packard. Each eight-pin mini-DIP houses one coupler. The input leads are on one side of the chip (pins 1 to 4), while the outputs are on the other side (pins 5 to 8). The chip can operate with a potential difference of up to 2500 volts between the two sides! There are six of these chips on the board, one for each control line.

The Sprague 2814 interface chips get their inputs from four CD4051 3-to-8 decoder chips. Address lines A0, A1, and A2 will select one of the eight output lines. The CD4051 also has an inhibit input. Only if the inhibit is released will any output from that chip become active.

A fifth CD4051 is used to select which one, if any, of the four main decoders will be enabled. Address line A3 goes to the low order input of this fifth decoder and selects between circuits 0..7 and 8..15. The fifth address line, which I call the ON/OFF line, selects between the drivers that connect to ON relay windings and those that connect to OFF windings. Finally, the inhibit input is controlled by the sixth addressing line. The microcomputer puts a 0.1 second pulse on this line after the correct driver has been selected by the other address lines.

Interfacing to the Heating System

The interface to the heating system controls is also made using GE low-voltage relays. In this case, however, the relays are being used as general computer-controlled switches and not for the direct control of 110 volt AC.

I will start with a description of the control of the thermostat setting. One's first thought would be to have a GE relay

directly turn the circulator pumps on and off. But what would happen, then, if the computer failed and the circulators did not go off or on? The house could easily overheat or freeze. And what would I do if the computer had to be shut down for repair?

For safety and reliability reasons, I decided to take an indirect approach. I installed a second standard thermostat. The original thermostat was left exactly as it was, but it was set to the lowest temperature to which we would ever want the house to go (about 50 degrees Fahrenheit). If the air temperature drops below that setting, the circulator pump will come on quite independently of the computer, and if the computer ever has to be shut down, all I have to do is go back to using this thermostat in the usual way.

The second thermostat is installed in parallel with the original one, except that a GE relay is installed in series with its wires. This thermostat is set to the highest temperature we would ever want the house to reach (about 73 degrees). Now consider a normal situation where we are trying to control the temperature at, say, 68. In this case, the first thermostat's circuit will be open (since the air temperature is above its setting), but the second thermostat's circuit will be closed (since the temperature is below its setting). Now, if the computer closes the GE relay, the circulator will run, and, if it opens the relay, the circulator will stop. Thus the computer can control the circulator, but the range of control is bounded by the settings on the two thermostats. If we set both thermostats to the same temperature, then we are back to normal control of the circulator.

Now let's look at how I control the temperature of the water in the boiler. I explained last time that the system has what is called an aquastat. Just as a thermostat senses the air temperature and opens or closes a contact, the aquastat senses the water temperature and opens or closes a contact.

I could have let the computer control the oil burner directly by installing a GE relay in place of the aquastat, but this would be risky. What if the computer made a mistake and failed to turn off the boiler? We could end up with an explosion. There is a second, safety aquastat that is supposed to protect against this, but I really did not want to rely on it.

Instead of removing the aquastat, I just installed the GE relay in series with it. In this way, the computer could turn off the oil burner, but it could not force the burner to remain on after the water temperature reached the aquastat setting. We set the aquastat to the highest temperature that would be needed on the coldest expected day.

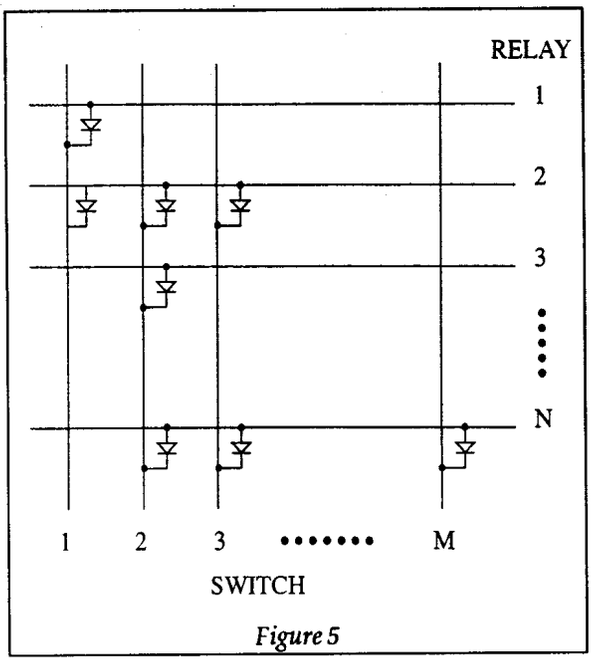


Figure 5

Now what happens if the computer fails to turn on the boiler on a very cold day? Our fail-safe thermostat arrangement will start the circulator pump, but this will not help if the boiler is off. To protect against this possibility, I installed another thermostat in the basement and wired its contacts in parallel with the GE relay. I had observed that enough heat leaked into the basement from the boiler that the temperature there never dropped below about 50 degrees, so I set this thermostat to just under that. Now if the computer should fail to turn on the boiler, this thermostat would short out the GE relay and again allow the regular aquastat to regulate the boiler. If the computer has to be removed for repair, all I have to do is raise the setting on that thermostat to, say, 80. Then the relay will always be bypassed, and the normal controls will operate.

The Sensors

As I discussed last time, the controller requires various environmental sensors. Management of the heating system requires knowing three temperatures: the temperature of the air in the house, the temperature of the air outside, and the temperature of the water in the boiler. I chose to use precision linear thermistors manufactured by YSI, the Yellow Spring Instruments Company. These are tiny modules, perhaps a millimeter across, that look like little capacitors. Inside there are two thermistors. When wired up with a pair of precision resistors, the circuit can produce either a resistance or a voltage that is a very highly linear function of temperature. Of course, the computer could have converted the signals from single, nonlinear thermistors to accurate temperature values, but it seemed much easier simply to buy the better components.

The thermistors come in various types for different temperature ranges. For example, the one for sensing inside air temperature is designed for temperatures from 30 to 100 degrees Fahrenheit. It has an absolute accuracy of plus or minus 0.3 degrees. I wired the thermistors for voltage outputs and then fed the signals to CMOS operational amplifiers. The gains of the amplifiers were set to provide a convenient mapping of temperature onto a one-byte binary scale in the output from the analog-to-digital (A/D) converter. For example, the inside air temperature uses 255 units of 0.1 degree Celsius to read from 0 to 25.5 (78 F). The boiler temperature, which spans the greatest range, uses 255 units of 0.5 degree to cover temperatures from 0 to 127.5. [By the way, with this project I made the conscious decision to go metric. All temperature displays and settings are in Celsius.]

The one other environmental sensor is a photocell to measure outdoor light level. I just grabbed some old photocells from a junk barrel and tested them until I found one with a logarithmic voltage response. This signal, too, goes to a CMOS op amp, whose gain is set to give a nearly full-scale input to the A/D converter on a clear, sunny day.

Closing Comments

Some of you may wonder how I managed to retrofit the GE low-voltage

See Home Control, page 21

Getting Started in Assemble Language

Part 2

By A. E. Hawley

In part 1, we discussed the background information you should have, the software tools you need, and the nature of assembler instructions used in AL programming. Boolean algebra was mentioned, and some references given. The concepts of boolean logic are so powerful and useful that another good exposition of it is recommended: that of Maley and Earle (reference 11).

In part 2,we will explore your computers memory space, emphasizing the relation between important addresses and a conceptual model. We will cover some basics of your operating system and how your program uses it. After a brief review of what Assemblers, Hex Loaders, and Linkers do, we will illustrate with a simple program some of the concepts and programming practices of effective AL programming. Finally, we will take a look at how to assemble and link a sophisticated PD program that you can use as the basis for a simple Local Area Network.

Memory Model

Programs you write will execute within a memory environment that includes other code: the Operating System. Such programs interact with the operating system, or at least avoid trashing it. It helps in writing programs and understanding other peoples code to have a mental image for guidance. Such an image is a Memory Model. *Figure 1* shows three such models. Here, the memory space represented on each line is the full 64K address space. Addresses increase from left to right, starting at 0. This technique can be used to outline memory usage in a more restricted range like your programs usage of the TPA space.

Three different operating system configurations are shown in *Figure 1*. The Console Command Processor symbol is shown as [CCP]. The brackets are used to indicate that the CCP is transient; it is replaced each time the Warm Boot entry of BIOS is invoked. Notice the two names given to the BIOS segment in the CP/M and ZCPR models. DRI originally distinguished BIOS and CBIOS. Actual implementations, however, have combined both into one module and called it BIOS. Here's a useful definition that is consistent with both approaches. BIOS is characterized by a set of at least 17 jump instructions (a jump vector) whose arguments are pointers to routines that perform functions as defined in CP/M and Z-system standards. The CBIOS contains a similar jump vector whose targets are within the CBIOS module. When BIOS and CBIOS are separate, the target routines may be in either module. The pointer for routines not in the BIOS points to the corresponding CBIOS vectors. When BIOS and CBIOS are one and the same, only those vectors that point to actual routines are retained. In *figure 1*, the models for CP/M and ZCPR3 indicate a unified BIOS structure. For the NZCOM system configuration, BIOS and CBIOS are separated. In the simplest case, only the warm boot jump in BIOS points to a routine in the BIOS module; all others point to the corresponding CBIOS jump.

```
|-P0-|-TPA----------------------|[CCP]|-DOS-|-BIOS=CBIOS-|
                                  CP/M

|-P0-|-TPA--------------|[CCP]|-DOS-|-BIOS=CBIOS-|-OTHER-|
                          ZCPR3x

|-P0-|-TPA------------|[CCP]|-DOS-|-BIOS-|-OTHER-|-CBIOS-|
                        NZCOM

P0    = Page 0,        TPA = Transient Program Area
CCP   = Console Command Processor (CP/M CCP or ZCPR3x)
DOS   = CP/M BDOS or ZRDOS or ZSDOS or ZDDOS
OTHER = other system modules or unused memory space
```

Figure 1–Memory Models

The space occupied by 'OTHER' includes the ZCPR3 system modules, such as the ENV, RCP, FCP, NDR, IOP, and several ZCPR buffers.

Let's put some numbers to some of the locations shown symbolically in *figure 1*. Sizes for page 0, CCP, and DOS are unambiguously defined for CP/M compatible systems. They are shown in *figure 2*. Whenever a Cold or Warm Boot occurs, it is the responsibility of the BIOS to install the address of the warm boot jump vector and the entry point to DOS in the base page at locations 1 and 6, respectively. Thus, your program can always calculate the top of TPA by using the relation show as the size of TPA: the highest address in the TPA is (0006)-7. Some programs need to make calls to BIOS functions directly; the address of BIOS is calculated as shown in the figure. Since each jump instruction is 3 bytes long, the address for the 4th bios function would

---

A. E. (Al) Hawley started out as a Physical Chemist with a side line love of electronics when it was still analog. He helped develop printed circuit technology, and contributed to several early space and satellite projects. His computer experience started with a Dartmouth Time-Share system in BASIC, FORTRAN, and ALGOL. His first assembly language program was the REVAS disassembler, written for a home-brew clone of the Altair computer. As a member of the ZCPR3 team, he helped develop ZCPR33 and became sysop of Z-Node #2. He has contributed to many of the ZCPR utilities, and written several. He is author of the ZMAC assembler, ZML linker, and the popular ZCNFG utility.

be BIOS+4*3.

There is no entry for the length of BIOS in *figure* 2. The BIOS, CBIOS, and other modules are as long as they need to be; there is no standard length! That's why absolute addresses cannot be given for DOS and CCP.

You can easily get these addresses for your system; run the public domain programs Z3LOC (for Z-System) or CPMLOCS (for CP/M systems). The ZCPR3 Peek command can show the data in your base page.

The sizes and addresses are shown in hexadecimal, as you have no doubt surmised. Sizes are also sometimes stated in Pages and in CP/M Records (usually shortened to just 'records'). One record is 80h (128 decimal) bytes. A Page is 2 records. The length of the CCP is 16 records; that of the DOS

```
SEGMENT         SIZE            ADDRESS

Page 0          0100            0000
TPA *           (0006)-00FA     0100
CCP             0800            DOS  - 0800
DOS             0E00            BIOS - 0E00
BIOS            ----            (0001)-3

* (0006)-00FA == subtract 00FA from the word at address
0006
This is the maximum TPA size, assuming CCP is destroyed.
```

Figure 2. CP/M & Z-System Constants

is 28 records. Sizes are frequently stated in a mixed notation; 400h (1024 decimal) is 1K. You knew that, didn't you? I'll never mention it again.

Did you notice that I did not suggest using a debugger to look at the values in the base page? Try it now, and compare the bytes at address 6 & 7 with those observed without the debugger. They're different, aren't they? Now look at figure 3, which shows what happens to the size of the TPA for several situations. Debuggers and RSXes modify the address at 0006 to point to themselves! Well behaved programs will use the address at 0006 to calculate the top of usable memory and will only write memory above that point at the risk of causing a system malfunction. Thus the debugger/RSX code is protected as if it were DOS!

Note that when an RSX is present, the CCP is also 'protected'; the space allocated to the CCP can no longer be usurped by the program.

When a program reads or writes to a file, it does so through calls to DOS functions. Each call transfers one record. When CP/M was first conceived, floppy disks conformed to an existing IBM standard; there were 128 bytes per sector. That's one record. Early CP/M documentation used the words 'record' and 'sector' interchangeably. But why 80h bytes, when 100h seems so much more natural? Could it be that in Octal, the same number of bytes is expressed as 10?

The Operating System Interface

Programs that you write will almost always require the services of the DOS and occasionally the BIOS. The services provided by DOS include all Input or Output (I/O) involving your Console, Printer, and Disk Drives. BIOS services may be required for such tasks as setting or reading a system clock, or accessing unique-to-your system I/O ports such as a Modem or Controller board. BIOS services are required by certain programs that must access the system tracks, because DOS functions have no way of specifying tracks below the directory storage area.

DOS functions are defined and described in the CP/M Operating System Manual from Digital Research, available from Elliam Associates (see "sources".) This is a valuable source of other system data structure definitions you will need. If you are fortunate enough to own ZSDOS/ZDDOS, then you have an even more informative manual. (also see "sources" to get ZSDOS) Your program invokes a DOS function by loading the function number into register C, loading any required argument into the DE register pair, and then executing a CALL 0005h instruction.

We observed above that the argument of the jump at 0005 points to top-of-TPA + 6, and not reliably to DOS. So how can a call to 0005 execute a DOS function? It is the responsibility of the Debugger or RSX to ensure that there is a chain of jumps that finally terminate at the DOS function! So the jump instruction at location 5 serves two purposes: it marks the top of memory assuming that the CCP is not there, and at the same time it is *the* place to call from a program to execute DOS functions. The entire point of this discussion is to firmly establish that (a) you must never change the bytes at locations 1 & 2, almost never change those at 6 & 7, and (b) you may use the address at 1 to find the BIOS and the address at location 6 to find the current top of usable memory.

We'll talk later about a useful program (ZREMOTE03) that *does* change the address at location 6.

The Assembly Process

Assembler Input—Main Source, Included Files

Your assembler takes as input a single source file that has been prepared with a text editor. Within that source file may

```
|-P0-|-TPA-------------------------|-DOS, etc.----------|
                                Maximum TPA

|-P0-|-TPA---------------|-DEBUGGER-|-DOS, etc.----------|
                      TPA Reduced by Debugger

|-P0-|-TPA---------------|-RSX-|-CCP-|-DOS, etc.----------|
                            TPA Reduced by RSX
```

Figure 3. Available TPA

be an INCLUDE or MACLIB instruction which causes another named file to be inserted into the input stream and processed just as if it had been part of the original source file. The main source file for ZCPR34 contains little more than a series of INCLUDE files! ZMAC permits nesting of included files. Other assemblers may not be as accommodating. I often use SYSDEF.LIB as an included file. You'll see it in *listing 1.* The source file is processed to produce (at your discretion) several kinds of output files, console displays, and printer listings.

Assembler Output—PRN, SYM Files and Console Displays:

The console displays and printer listings are for your own use in following the course of the assembly. The same listings may also be written to files. The .PRN listing contains the code to which the instructions in the source file were translated, along with a copy of each source line (even those that produced no code, like blocks of comments). A

symbol listing may be produced as a separate .SYM file or appended to the .PRN file. Command line options select which, if any, of these are created.

Assembler Output—HEX, REL Files:

One of two kinds of file may be produced for use by another program. A HEX file may be produced for use by LOAD, MLOAD, or MYLOAD. HEX files may also be used by EPROM or PROP programming software/hardware. A REL file is used by a Linker and by linking loaders like NZCOM and JETLDR. Which of these is produced is your choice when you run the assembler; an optional argument on the command line tells the assembler which kind of file to produce. For example the following command line is used with ZMAC to produce a HEX output file:

```
ZMAC MYFILE H
```

Without the 'H,' ZMAC defaults to production of a REL output file. The SLR Z80 and HD64180 assemblers produce either standard Microsoft REL files or SLR-REL files, selectable with a command line switch. Use the Microsoft REL output if you want your REL files to be usable by others who may not have SLR! All linkers can use mREL files.

Assembler Output—COM Files:

SLR assemblers for Z80 and HD64180 can produce a COM file directly. Producing COM files directly is very handy for small programs, but suffers from lack of flexibility afforded by the two-step process of assemble-and-link. If you have an SLR assembler, by all means take advantage of its capability to make COM files directly. But beware of stopping there! Many of the most useful features of AL programming are a direct result of the two step process. HLL programmers are generally aware that their function libraries are made possible by one step of the compilation process: LINKING.

HEX Loaders

The first widely used assembler for microcomputers was ASM, provided with CP/M by Digital Research. ASM could make only HEX files, so LOAD was used to convert the HEX files into COM files. HEX files contain a series of data records (note: *not* CP/M records!), each of which comprises a starting address, a byte count, and the bytes to be loaded at successive memory locations. The loader does the obvious: it deposits the bytes in memory at the indicated addresses. The public domain program MLOAD by Ron Fowler completely replaced LOAD, because it is able to load code at locations other than starting at 100h. MLOAD can also perform another vital function. It can combine an existing COM file with the code loaded from a HEX file, overwriting the code at the designated addresses in the COM file. This is a popular method of installing configuration data in already-linked COM files. Under CP/M, this was a preferred method of installing the data necessary to adapt the program to your particular terminal for screen oriented programs. Take a tip from Digital Research: they did *not* use ASM (or MAC) and LOAD to produce CP/M and its utilities!

The Linking Process

The job of the linker is to produce a file which can be loaded into memory and executed by your computers cpu. To do this, The linker accepts one or more files as input. It interprets and combines the code from each to make a single binary output file. The rules which the linker follows during this process come from three places: from algorithms built into the linker, from instructions encoded in the source file(s), and from command line options that *you* specify.

Linker Input and Output files:

REL files as produced by the assembler contain special code sequences at the start and end of each file. The linker detects these tags, so it knows the limits of each module independently of the actual file contents. Because of this feature of REL files, it is possible to concatenate any number of REL files to make one Library file which may be named as an input file for the linker. SYSLIB, VLIB, and Z3LIB are such REL libraries. Note that these REL libraries have nothing in common with the popular .LBR type libraries! They are in no way interchangeable. The linker extracts and links only those library members that are needed by your main REL files (external references).

If you have ZML, then there are two additional input files that you can specify as linker input. They are the PRL and RSX header files. These files contain normal code, produced perhaps in a separate linking operation. Here's the order of the components in a ZML binary output file:

```
[RSX_HDR] [PRL_HDR] (CODE_IMAGE) [BIT_MAP if PRL]
```

The HDR files are included only if requested by an option on the command line, and the Bit Map is only generated if a PRL file is being constructed. Bytes at offset 1 and 2 in the PRL HDR will contain the length of the code image that follows. ZML puts it there at the same time that it produces the BIT MAP following the code image. In PRL mode, ZML makes a null filled (except bytes 1 and 2) PRL header unless you give it the name of a file which you want used. The PRL header file specified usually contains code which uses the length word and the bit map to move the image into its final destination in memory and then adjust address references in the relocated image to correspond with its memory location. Type 4 programs used with ZCPR34 are PRL files whose PRL header file is publicly available as T4LDR. T4LDR contains the code that performs the module relocation. We'll discuss RSX type headers later in connection with ZREMOTE03, which uses both RSX and PRL headers. RSX type programs are discussed by Bridger Mitchell, who defined the standard structure of such files for CP/M 2.2 and Z-systems (reference 12). Hal Bower (reference 13) describes an RSX program and how you can build it without ZML.

Linkers can produce a SYMbol file, which contains program symbols and their final linked addresses. Only those symbols originally declared PUBLIC (or one of its synonyms) in the original assembler source file are included in this listing, because those are the only ones whose name is passed in the REL files.

Writing a Program

That's enough introduction, especially for those who have digested (in-digested?) the material in the references! First we'll state the circumstances that called for this program, shown in *listing 1*.

My Z-Node runs on an aging S100 computer. It is so old that the battery that runs the RT Clock has died! Other more serious ailments cause the system to crash at random times.

It needs an extended time on the workbench. So, I have put together a new system using an Ampro Little Board Plus. I need to transfer about 30MB of files from the S100 to the new machine. The S100 only talks to 8" disks, and the Ampro only talks to 5 1/4" disks! I could transfer files via modem, but that's awfully slow at 2400 baud. ZREMOTE to the rescue! ZREMOTE (ZR) is BYE stripped down to the bare essentials for direct RS232 connection and implemented as an RSX. With MEX running in the Ampro and ZR in the S100, a trial run showed that I could transfer files at 9600 baud. KMD runs just fine under ZR. But the remote terminal dropped characters. The solution was to modify ZR to provide a delay (equivalent to nulls) after each line transmitted to the remote. That delay was proportional to the the number in the ZR data pool that specifies added nulls. I needed an utility that could change the number of nulls while ZR was installed in high memory. The program I wrote is *listing 1*, SETNULL.

When ZR is running, it intercepts BDOS calls and tests the value being passed in the C register. If that function number is one that belongs to BDOS, it is passed on. If it is one of the functions supported by ZR then the function is performed by a subroutine in ZR before returning to the caller. Function 72 sets or returns the number of nulls, according to the value passed in the E register. The signal to return the current number of nulls in E is 0FFh. Any other value is the number of nulls to set. BYE works the same way.

Here are some specifications for the program:

1. If ZR is not running, the program should do nothing.
2. If the program is invoked as SETNULL / or as SETNULL // then a help screen should be displayed.
3. To set the number of nulls to 15 the command is SETNULL 15
4. If there are no arguments, then the program should set nulls to some default value. I chose 16 as the default, because that was the number that made the remote behave properly.
5. Except for the help screen, there is to be no console output. (This is a personal preference. I don't like programs to 'chatter'.)

The first part of the program is a set of standard comments showing the program identification, date, and function. A year from now, I won't have any trouble knowing what this code is all about!

I save some time by using a set of standard symbol definitions in SYSDEF.LIB (in the public domain). For example, CR is defined there as 0Dh, YES is defined as 0FFh, NO is defined as 00, and so on for commonly used symbols that convey meaning better than raw numbers.

The program will be parsing the command tail for a number, and it will be printing to the console for HELP (//), so we'll save some programming time (and possible errors) by using those routines from SYSLIB. The .request pseudo-op puts the name of the library to search for those routines into the REL file, where the linker will find it and know to search the REL library.

I use a standard skeletal form for programs with spaces allocated for this boilerplate. The form also includes the first part of the code up to and including the label START:. It also includes the line labeled EXIT: and the 'END' pseudo-op at the very end of the listing. Everything else gets inserted in a copy of the form with my editor. Here, the edited file gets named SETNULL.Z80.

The main routine at START is short, a very desirable goal for any program. Since this program is short and simple stack usage is very modest; it is not really necessary to set up a local stack. The code involving the SP here, in the exit routine, and at the program end could be deleted. I left it in because it is an excellent habit to cultivate; stack overflow causes mysterious or catastrophic behavior in a program. *After* the program is working you can reduce stack size (or eliminate the local stack) if you need to save a few bytes.

Note the EXIT: routine. When CCP loads this program, it transfers control to the program with a CALL 100h. Thus, a simple RET is all that is needed after restoring the CCP stack to get back to the system prompt level. If the program were large (or greedy) enough to over-write the CCP, then a RET would result in a system crash. In such a case the proper way to exit would be via JP 0, which results in a warm boot and restoration of the CCP. These cautions illustrate a fundamental responsibility of the AL programmer that is far less visible in the HLL world: memory usage and stack usage must always be considered.

The coding of SETNULL is straightforward, and copiously commented. Coding practices in AL follow about the same rules as for HLL; for a refresher on coding practices I recommend owning and reading a copy of Kernighan & Plauger (reference 15). You will observe in studying the listing another principle: your program will often depend on a knowledge of data structures that are not part of the program itself. The extended BDOS calls that ZREMOTE makes available are an example; the test for the existence of BYE/ZREMOTE is another. SETNULL can be assembled with ZMAC, M80, and Z80ASM or SLR180. It can be linked with any of the companion linkers for those assemblers to make a .COM file which, when executed, will set the default number of nulls of a running BYE or ZREMOTE program. Here is the assembly/link syntax for ZMAC/ZML (with SYSLIB.REL present on your current directory):

```
ZMAC    SETNULL
ZML     SETNULL
```

Refer now to *Listing 1*.

ZREMOTE

ZREMOTE is a program that performs the essential functions of BYE, essentially putting your Console and another I/O port on your computer in parallel. To that port may be connected another terminal or another computer, through the normal RS232 connectors. ZREMOTE is an RSX which loads into memory just below CCP/ZCPR and stays there until it is removed by Bridger Mitchell's REMOVE utility or you do a COLD start of the machine. If you have KMD installed, and the remote computer has a communications program like MEX or IMP, then you can transfer files and communicate between the two computers just as if the ZREMOTE machine were a Z-Node and the machine with MEX is a caller. No modem is required. You can set the baud rate to much higher than the usual 2400 baud maximum with modems for very fast file transfers. ZREMOTE is available on Z-nodes. I won't say much about how the program works because this article is already too long. I want to show, though, how the program can be assembled simply and efficiently with ZMAC and ZML.

See Assembler, page 48

Local Area Networks

By Wayne Sung

LAN stands for Local Area Network, a way of connecting computers together to allow information to be sent among them. The 'local area' part of the name is meant to imply a small distance between the farthest stations on the network, for example adjacent buildings.

Two rules seem to be true of local area networks. The first is that each individual network grows larger and at a faster rate than would have been expected. The second is that more and more often it becomes necessary to join multiple networks together. Both of these situations require good engineering when building networks.

Every LAN system has defined limits for things such as cable lengths. In the beginning stages of a LAN most of the limits can be stretched. However, as the network grows, there will be mysterious failures that in the worst cases require total reworking of the wiring.

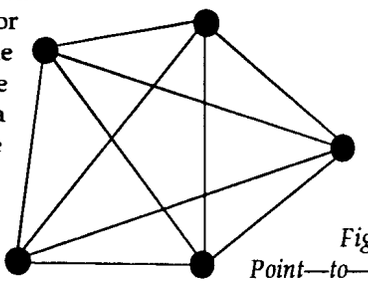


Figure 1
Point—to—point

There has been much debate about what kind of LAN is the best. In many of these kinds of arguments, although some of the facts and figures are correct, the overall picture is clouded by vested interests. In reality, the LAN hardware is probably the least important part of the total system.

What I propose to do here is to develop from more fundamental principles how a LAN works and perhaps show that there is really not that much difference between one type of LAN and the next. As we go along, I will give examples primarily based on Ethernet since this is what I work with the most.

Ethernet embodies some techniques that are often misunderstood, so it would be well to have them explained. However, any correctly designed LAN behaves well and becomes invisible to the user. Unfortunately, there are many bad habits that LAN designers have gotten into that make tuning a LAN difficult. We will get into all that.

Let's look at hooking up computers in general. A small number of computers can be connected with a number of point-to-point lines (*Figure 1*). The key word here is small, because if every machine is to have a direct connection to every other, then we find the total number of links goes up by the square of the number of machines, L = N * (N-1). Typically, machines are not equipped to handle very many ports. Also, there must be tables that define who is connected to which port.

An alternative to this arrangement is to let some machines have lots of ports and others have only a few, or even one (*Figure 2*). This arrangement, often called tandem switching, allows certain machines to handle switching for others. Each machine, especially the ones with many links, must still know who is potentially reachable by which link.

The biggest problem here is that if one of the intermediate machines fails, many machines lose contact with each other. Also, with an indirect connection, all transmissions must contain information to identify the source and destination of a message.

Another possible arrangement is the bus LAN (*Figure 3*). All units are connected to a party line. Since the tandem arrangement already requires source and destination identifiers, we can use hardware to recognize these identifiers. In essence we take the switching machines and distribute them among individual computers (the networking hardware).

In addition, we raise the speed of the line significantly to accommodate the total number of machines. This is the same idea as having a very fast computer divided among many users so that each appears to have a moderately fast one. Since we have distributed the job of switching to the network hardware in each computer, a higher speed link can be provided by this same hardware.

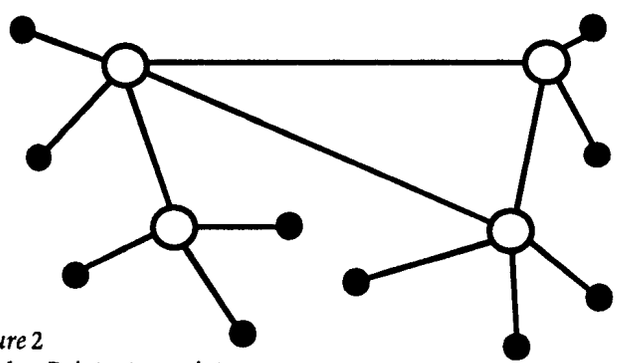


Figure 2
Tandem Point—to—point

One of the most common misconceptions is that the signaling speed of a LAN has something to do with the

Wayne Sung has been working with microprocessor hardware and software for over ten years. His job involves pushing the limits of networking hardware in attempting to gain as much performance as possible. In the last three years he has developed the Gag-a-matic series of testers, which are meant to see if manufacturers meet their specs.

processing ability of the computers using that LAN. In fact, the high speed is there only so that more units are able to use the LAN in turn, not so that any one unit can have it all.

A fundamental assumption of LAN technology is that the traffic on it is uniformly distributed. If there is one large machine that many small machines want to get to, this is probably the worst kind of arrangement for a LAN because the traffic distribution is not at all uniform.

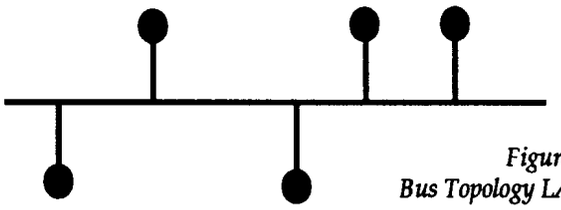


Figure 3
Bus Topology LAN

A big difference between a centralized switch and a distributed switch (which is what a LAN is) is the hardware broadcast mechanism available. With all units attached to a party line, any information to be made known to all units can be broadcasted. This is no more work than an ordinary message, since all stations hear all messages anyway. In the tandem switches, broadcast messages must be replicated and sent to each port separately.

It is also possible to form subsets of broadcast messages called multicast messages. These are messages that many, but not all, units might be interested in. Indeed, broadcast is a special case of a multicast. The networking hardware is programmed to recognize these special addresses, in addition to some address that it is explicitly known by.

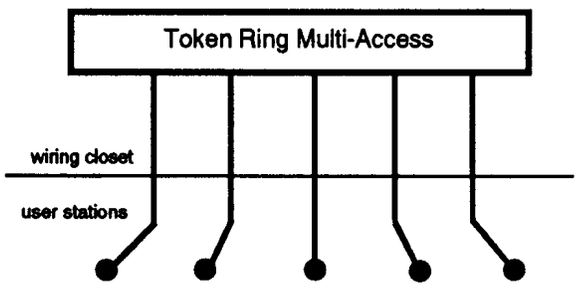


Figure 5 Ring wired as Physical Star

In the LAN case, then, the problem of maintaining a table of who may be reached by the network port reduces to one of "they're all out this port". It's really not quite that simple, but the broadcast mechanism makes it almost trivial. One unit sends out a broadcast that asks whether a certain other unit is available. If so, the second unit sends a message back to the first one and any further messages occur directly between those two units.

Unfortunately, the broadcast mechanism is also very easy to abuse. Many systems routinely send out broadcasts in the spirit of "I thought you might like to know". This is a reasonable assumption where only one type of software is in use. This is not usually true with LANs.

Where many different types of systems are in use, not all systems have to work with all others. In the case of a broadcast, though, there is no choice. By definition, everyone receives the broadcast and has to check it even if only to disregard it. One of the easiest ways to rob everyone on a LAN is to send a lot of broadcasts. The cpu cycles required to process the broadcast come out of everyone's machines.

There is also much confusion between the networking system itself and the shape of the physical network. A number of 'topologies', which refers to the way switching is done, are common.

In a star topology, several units are attached to a central unit, which handles all the switching. In a bus topology, all units attach to the same wire (much like a backplane bus), and each station can hear transmissions from all other stations, including itself. The station does not process any messages unless the specific station address is in the message.

In a ring topology, each station is attached to one upstream and one downstream neighbor. Messages always flow downstream, and, again, whether it is processed or not depends on whether the specific station address is present.

Totally separate from this logical topology is the physical topology, i.e. how the wires are actually run. Ethernet, for example, is a logical bus. However, in many installations, it is more convenient to run a number of stations back to one central spot (for example, a wiring closet). This makes a physical star (*Figure 4*).

By the same token (oops!) a token ring system is obviously a logical ring but is also most often wired as a physical star (*Figure 5*). In essence, the physical arrangement and the logical arrangement are totally separate, and any logical arrangement can run on any physical arrangement.

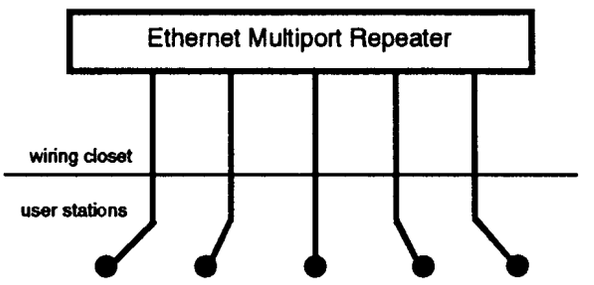


Figure 4 Bus wired as Physical Star

There is a further confusion. There is a combination of hardware and software to allow stations on a network to transmit messages to each other. It is a totally separate issue what services are available in a system of computers, whether networked by way of a LAN or by way of a large multi-user mainframe.

It is not at all necessary to have a LAN to be able to have, for example, mail or file services. In fact, using a LAN for centralized services almost completely negates the purpose of one. Still, in small systems you can get away with it.

Consider this: if you are sending a message, you have the whole message in hand from the beginning and can give it to the networking hardware all at once. The networking hardware will send it as soon as possible and will send it at the signaling rate of the LAN.

On the other hand, if you are to receive a message, you don't start off

See LAN, page 52

The Z-System Corner

PCED, the Z-System for MS-DOS Computers

By Jay Sage

The main subject for this column will be PCED, an operating system extension program for MS-DOS that gives one a working environment as close to Z-System as I have seen for MS-DOS. First, I have a few other small items to cover.

ZEX Script Correction

My ZEX script for formatting disks on the SB180 using the FVCD utility had a mistake, and I would like to publish a correction (due once again, I believe, to the legendary mistake-catcher and bug-fixer Howard Goldstein).

I, and many others, keep forgetting that the flow-control string tests performed by the IF commands (resident and transient) are not string tests at all; they are file name tests. Thus, if we have

```
IF EQ D:A.B B:A.B
```

the test will return TRUE, since the file name parts are identical. Directory prefixes are ignored.

In my ZEX script I wanted to detect a string of either "F" or "F:". The proper way to do this is with the command

```
IF EQ :$1 F
```

The colon before "$1" forces the "$1" string to be taken as a file name even if it contains a colon, since a directory prefix (the colon) has already appeared. Any additional colon in the "$1" string will be taken as a file name terminator, and it and anything after it will be ignored. Thus, if the command line token is "F:JUNK", the test will return TRUE.

Z-Node Update

It has been some time since we last published the Z-Node list, and there have been a couple of changes. We lost Z-Node #73 in Missouri. On the other hand, Z-Node #40 in Winnepeg, Manitoba, has been revived after years of neglect. The node has been transferred to new sysop Greg Kopp. We also have good news about accessing nodes 21 and 32 in South Plainfield, NJ. Both can now be reached by PC-Pursuit.

Finally, I would like to pass on a message I received by Internet email from Helmut Jungkunz, the incredibly energetic Z-System enthusiast in Munich, Germany. (He has produced German editions of the NZCOM and Z3PLUS manuals!) See the side panel for the text of his message.

PCED: Professional Command Line Editor

We now turn to the main subject for this issue. For many years, even after I had a Compaq 386/16 DOS computer at work, I continued to use CP/M machines. Among my collection were a couple of Ampros and a Wave Mate Bullet. The Z-System was so far superior to MS-DOS that the CP/M machine was often easier to use, even though its raw computing power was far less than that of the DOS machine. Several circumstances have led me at this point to retire the CP/M computers at work.

For one thing, I make heavy use of several applications that require a big machine. I do all my writing, including scientific and mathematical work, with TeX (actually LaTeX); I do a lot of my calculating with MathCAD, a computerized blackboard; and I now use PSPICE for electronic circuit simulations. All of these applications require about 500K of free memory. This is way beyond anything CP/M could dream of supporting; unfortunately, it sometimes exceeds what MS-DOS can provide! I have to be very careful about the drivers I use and how I load them, or I end up with too little free memory. The 386 is now tweaked up to the point where it just barely handles the jobs I need done.

The second thing is an operating system extension called PCED (for Professional Command Line Editor) that makes DOS bearable. It provides the closest thing to a Z-System interface that I have found for MS-DOS computers. I began to use version 1 of PCED

Jay Sage has been an avid ZCPR proponent since the very first version appeared. He is best known as the author of the latest versions 3.3 and 3.4 of the ZCPR command processor, his ARUNZ alias processor and ZFILER, a "point-and-shoot" shell.

When Echelon announced its plan to set up a network of remote access computer systems to support ZCPR3, Jay volunteered immediately. He has been running Z-Node #3 for more than five years and can be reached there electronically at 617-965-7259 (MABOS on PC Pursuit, 8796 on Starlink, pw=DDT). He can also be reached by voice at 617-965-3552 (between 11 p.m. and midnight is a good time to find him at home) or by mail at 1435 Centre Street, Newton Centre, MA 02159. Jay is now the Z-System sysop for the GEnie CP/M Roundtable and can be contacted as JAY.SAGE via GEnie mail, or chatted with live at the Wednesday real-time conferences (10 p.m. Eastern time).

In real life, Jay is a physicist at MIT, where is tries to invent devices and circuits that use analog computation to solve problems in signal, image and information processing. His recent interests include artificial neural networks and superconducting electronics. He can be reached at work via Internet as SAGE@LL.MIT.EDU.

many years ago. It was a great improvement over plain DOS, but there were many areas where the author just had not thought things through far enough. This, I find, is typical in the DOS world, where programmers do not share their work as we do in the CP/M community, and any given program tends to be limited to what one programmer can accomplish.

I contacted the author of PCED several times, and during one call he told me that he was working on release 2. At that point, I began to lobby heavily for certain improvements. Most importantly, I sent him copies of old *TCJ* columns that described ARUNZ and LSH. PCED had alias processing, but the parameter parsing was as feeble as that provided for BAT files. Only complete tokens could be passed; there was no way to extract pieces of tokens, such as just the path or just the file name or extension. There was a command history facility, but it, too, was extremely primitive. To find an old command line, one had to back up line-by-line until the right command was found; the computer did not provide the help with searching the way EASE and LSH do.

Well, PCED release 2 is now out. The author has not copied the user interface of ARUNZ and LSH, but he has incorporated most of the functionality. In addition, with the far greater memory available on a DOS computer, PCED is able to do some things that we cannot with our 64K machines.

I'm sure I am not the only Z-System user who also has occasion to work on DOS machines, so I would like to use this column to describe some of the main features of PCED. Perhaps we can even learn something that we can apply in the Z-System. I will start with the features that are most like those we are familiar with from the Z-System and then go on to some of the extra capabilities of PCED.

As usual when I see a program that should be of great interest to Z-System enthusiasts, I try to get it added to the Sage Microsystems East product line. PCED is no exception, and SME now offers it for the very attractive price of only $50.

Multiple Commands on a Line

With PCED running you can enter multiple command lines at the DOS prompt just as you are used to doing with Z-System. You can set the command separator to the character of your choice. The default is the caret character, which I find to be a nuisance to type, since it requires pressing the shift key. Besides, for Z-System compatibility I prefer the semicolon. Occasionally this causes a problem, however, because some DOS commands use a semicolon in their syntax (the PATH command, for example).

There are several ways around this problem. There is a simple command to turn off PCED entirely so that user input goes straight to DOS. Then one can enter such commands with no interference. One can also change the separator character on the fly. Thus a single command line can change it to the caret, run the command that requires a semicolon in its argument line, and then switch it back to semicolon. Such a command line might look like this:

```
ced chainch ^;path dir1;dir2;dir3^ced
chainch ;^echo back to semicolon;echo all done
```

The first command, separated by a semicolon, is

```
ced chainch ^
```

CED.EXE is the executable program that constitutes PCED. It's 'chainch' option redefines what PCED calls its chain character (what we know as the command separator). PCED then parses off the next command using the new separator:

```
path dir1;dir2;dir3
```

PCED is able to do this because it buffers user command line strings and feeds the individual commands one at a time to DOS's COMMAND.COM. Since, unlike MS-DOS, the Z-System supports multiple commands on a line at the operating system level, ARUNZ ships the entire alias command line to the command line buffer immediately.

Alias Scripts

PCED supports alias scripts, which it calls 'synonyms', and it has much of the alias processing power of ARUNZ. As a result of our efforts, the author has incorporated a very powerful parameter parsing facility. It can pull apart tokens into the drive, path, filename, and filetype, and it can return the current date and time and the default drive and path. It lacks only the parameters for accessing memory and the system environment (things like the number of drives defined). PCED command lines can recognize DOS environment variables created using the SET command. As under the Z-System, aliases can be nested to any degree. The only limit is the total length of the multiple command line buffer. The default length is 512 bytes, but you can make it longer if you like.

Here is a command that defines a synonym that displays a directory listing with the file specification automatically wildcarded:

```
ced syn d "cdir %1{$r*.$e*} &r"
```

The 'syn' option on the CED command causes it to enter a new synonym (alias) definition into memory. In this case, the name of the alias is 'D' and its prototype command line is the text enclosed by delimiter characters (here the double quote, but any character may be used).

The script in this example invokes the CDIR command, which lists the files in a directory. We will have more to say about CDIR later, but any directory listing program could be used (even DOS's horrible DIR). What is interesting here is how parameters are parsed.

As with MS-DOS batch files, command line tokens are designated by the percent sign followed by a number. What is different here is that curly braces can be used to indicate that further processing is to be performed. The dollar sign introduces special symbols. The first one, '$r' represents everything in the token except for the file type and the dot separator. The form '$e' represents the extension. Other characters are literals to be included exactly as entered into the script.

There are parameters of a different type that are introduced by an ampersand. In this example '&r' represents the rest of the tokens on the command line after any that have been referred to explicitly (%1 in this example). Thus, if the user enters the command

```
d d:\main\sub\a.b /date
```

PCED will generate the DOS command

```
cdir d:\main\sub\a*.b* /date
```

A question mark can be used to define a default value for the

parameter in case the user enters no token. Here is an example:

```
ced syn format "format %1{?a:}"
```

Now when the user enters just "format", DOS will get the command

```
format a:
```

but if the user enters "format b:", that is what PCED will send to DOS.

If you have been really alert, you may have wondered how we could get away with defining an alias command named FORMAT in terms of a real command called FORMAT. Well, PCED is pretty sophisticated in the way it handles this and prevents infinite recursion.

First, it is important to note that PCED always treats commands as aliases first. This contrasts with the way ARUNZ operates under Z-System, where ARUNZ is invoked only if the command processor cannot handle the command. PCED's behavior arises out of necessity. In Z-System, ARUNZ is a tool invoked knowingly by the operating system; PCED, on the other hand, has to sneak in front of the highly incompetent MS-DOS operating system.

Second, PCED provides several ways to turn off special processing of a particular command. The command

```
ced ignore cmdname
```

will cause CED to completely ignore any command line that starts with the specified command. There will be no parameter expansion and no stacking in the history. This is the most extreme way to turn off special processing.

A slightly less drastic measure is to put a special character in front of the command that tells PCED to ignore the command this time only. The default for this character is the asterisk, but for compatibility with Z-System I redefine it to be the period using the command

```
ced ignorech .
```

Then when I enter a command like

```
.path subdir1;subdir2;subdir3
```

This, obviously, is another way to permit the command separator in the command line.

Finally, a command can be entered with a leading space, since synonyms are matched only if they begin with the first character. For example, if I have defined an alias with the name DIR but want to run the standard DOS DIR command, I can enter either of the following commands:

```
.dir
<space>dir
```

The latter has the advantage that the command line will still be processed by PCED and placed on the history stack. This is especially useful for an alias definition like the following:

```
ced syn dir "echo running dir; dir %1"
```

Without the leading space before the DIR in the script, this would be a circular definition, and any invocation of DIR would result in a command stack overflow in PCED. The space option must be used here if we want the parameter expansion to be performed. If the first command in a definition is the same as the synonym name, then synonym processing is automatically suspended for that command; that's why our FORMAT alias above was OK.

Well, I think this is enough detail to give you the idea of the sophistication that is possible with PCED aliases. One thing I would like to make clear at this point is that we do not have to enter all PCED commands (such as synonym definitions) manually each time we start the computer, as the examples above may have suggested. This would clearly be an enormous inconvenience. PCED includes a 'load' option that will read a file containing any valid PCED commands, that is, arguments that would normally follow a CED command. For example, I have a file called PCED.CFG that I invoke as part of my AUTOEXEC.BAT file using the following form:

```
ced load c:\pced\pced.cfg
```

The PCED.CFG file contains lines like the following:

```
chainch ;
ignorech .
syn d "cdir %1{$r*.$e*} &r"
... (other synonym definitions)
```

History Shell

PCED provides a command history shell environment like that of LSH. Command lines can be edited nicely as they are

Z-System Report from Germany

Today, the sysop of NBBS and I have rearranged the file section for the ZNODE. Still the ZNODE server will carry directly accessible menus and messages to experiment with ZCPR, even if people do not run ZCPR themselves. There is one SIG for all Z80, 64180, Z280, and even 8086 (or the like) CP/M computers, where I work as a SIG-op. The file section is broken down into:

1) CP/M general
2) CP/M 2.2 only
3) CP/M Plus approved or only
4) ZCPR
5) CP/M 86
6) Librarians (Pack/Unpack tools)
7) Editors and Text-Tools (Non-WS)
8) WordStar Trouble Shooting
9) Database Programming
10) Turbo Pascal
11) Assembler and 'C'
12) Information for CP/M
13) Information for ZCPR
14) The Computer Journal
15) I forget....

Anyway, something like this is what the new NBBS structure is going to look like. So much for people's questions about RCPMs in Germany. Also, there is a good one in Wuppertal, WODS Wuppertal Online Database Service. They have very good support for C128, MSX, and Spectravideo machines, but also for general CP/M. Maybe you can send part of this info on to other ZNODES. Hello and bye from Munich, Germany, Ciao, -> Helmut Jungkunz <-

entered, just as with LSH. The history function in PCED is called the command stack. Commands can be viewed either in line mode or, by loading the optional VSTACK program, in full-screen mode. If a partial command line is entered, the search function returns only those lines that begin with that string. Even the full-screen display shows only matching lines. Unlike LSH, however, one cannot edit the full command history but only individual command lines.

Another advantage LSH still has over PCED is in the options it offers as to how the next-command pointer is positioned after a command has been executed from the history. In PCED, the pointer always points between the command just executed and the one that followed it in the past. This is similar to LSH's auto line sequencing mode, except that LSH is actually ready to execute the next command line with just a carriage return. With PCED one must press the down arrow key.

I have found that PCED's method easily results in confusion as to which direction one should search for a new command. I have LSH configured to use line auto line sequencing only in full-screen mode but to put the pointer at the bottom of the history when in line mode. In general, I think that the author of PCED could have learned a lot had he been able to see LSH in operation.

Command Completion

PCED has a command completion feature similar to the one in LSH, but slightly more powerful even. As with LSH, if you begin to enter a command line, such as

```
edit tc
```

and then press the TAB key, PCED will try to complete the last token on the line, "tc", by looking up all files in the specified (or current) directory and displaying the next one each time the TAB key is pressed. Thus if the directory included TCJ49.WS and TCJ50.WS, the command line would cycle through the following as the TAB key is pressed:

```
edit tcj49.ws
edit tcj50.ws
edit tc
```

PCED also allows one to enter a wildcard file specification and then have the TAB key display the matching files in sequence. Thus one might enter the command line

```
edit *.ws
```

I thought this was something that LSH did not offer, but I just tried it and discovered, to my surprise, that LSH does this too!

The area where PCED is a little more advanced is with completion of the first token on a line. PCED is smart enough to scan only executable files and to omit the file type. It also automatically supplies a trailing space so that one is immediately ready to enter any command arguments. The closest thing to this with LSH would be to enter a command line of

```
e*.com
```

and then press TAB until "edit.com" appeared. Then one would backspace over the ".com" and enter the command tail. PCED does this automatically when just "e" is entered on the command line. PCED will also include in the scan PCED synonym command names. This is a nice feature, but I don't make much use of it, perhaps because I am not accustomed to it from Z-System.

Flow Control

PCED provides a very rudimentary flow control, vastly inferior to what we have in Z-System, but better than nothing. A CED option called CANCEL will discard any pending commands and insert the command (which may be an alias) given in its argument. It can be used with the standard DOS IF command, as in this example:

```
ced syn test "if ex %1{$r.bak} ced cancel
  dobak %1;nobak %1"
```

If a BAK file already exists, the DOBAK command or alias will run; otherwise the NOBAK command or alias will run. This is essentially a single level of flow control with no ENDIF. It reminds me of the so-called recursive alias that I invented before Dreas Nielsen showed how to do recursion rigorously under Z-System (see *TCJ* issues 27 and 28). If TEST is called from another alias with multiple commands, all pending commands from that script will be lost if the CANCEL branch is taken. This makes nesting impossible and can cause some unplanned, unexpected, and undesirable behavior.

With a little bit of work, I think that the author of PCED could have implemented a full flow facility like the one in Z-System. Having had no experience with it, he probably could not appreciate just how much power it provides. Perhaps I will be able to convince him to put it into version 3 of PCED.

Unique Features of PCED

While PCED does not have all the power of Z-System in the areas described above, it does have many features that we do not have, and probably could not have given our memory constraints.

PCED operates entirely from memory, with fixed-length buffers allocated for its various features. As a result, PCED performs its functions with lightning speed. Having its alias definitions and other information in memory also allows it to do some interesting and useful things, some of which we will describe below.

Dynamic Alias Definitions

As we have already seen, PCED allows new alias definitions to be added directly from the command line at any time. This can be handy in its own right, but what is really spectacular is that one alias can create or remove other alias definitions! For example, one can create an ADDPATH alias that not only adds a new directory element to the search path but also creates an alias RPATH (restore path) for resetting the path to its previous configuration. RPATH includes a command (the PCED "clear syn" option) to remove its own definition after it has performed its function!

There can be multiple definitions for the same alias, with the newest one always being used. Thus, running ADDPATH several times will add several new path elements and create several new RPATH aliases. Each time RPATH is run, it sets the path back one step and then removes itself. Thus, the PCED aliases create, in effect, a path stack.

Many other creative uses like this can be implemented. For example, I have some scripts like the following:

See Z-Corner, page 57

Using The ZCPR3 IOP

to Add Function Keys to a Kaypro 10

By Lindsay Haisley

A few years ago I dropped the keyboard of my Kaypro 10, broke a key, and had to remove the steel cover to make the necessary repair. While I had the cover off, I had an opportunity to examine the internal workings of the keyboard. The Kaypro keyboard, at least on my K-10, was manufactured by a company called Maxi-Switch. It is a general-purpose keyboard which is apparently capable of doing a lot more than Kaypro asked of it. What caught my eye right away was the presence of numerous switch-sized square holes in the metal grid which holds the keys in place, each with corresponding marks and holes for switch leads on the underlying pc board. A little experimentation revealed that all of these unused key positions were functional, and hardware implementation of them involved no more than soldering in key switches and cutting away the metal keyboard cover to make room for the added keys.

Software implementation of the four extra keys which I originally installed required a few minor modifications to the stock Kaypro BIOS. I submitted an article to Kaypro's Profiles magazine on the subject which was never published. I learned a great deal from the project, however, which led to the system improvements described in this article.

The keys on the Kaypro are of two types. The "normal" keyboard, consisting of the traditional typewriter keys, the line feed, the control keys, the escape, etc., all return the expected ASCII value. The number keypad keys, the cursor control keys, and the extra key positions which I discovered return values which have more to do with their physical position in the key layout than with their meaning or function. The defining characteristic of the latter group of keys is that they all return values with the high bit set, i.e., in excess of 7f hex. The stock Kaypro BIOS CONIN routine tests for this high bit when the keyboard is read and proceeds accordingly.

My Kaypro 10 has evolved over the years and now sports an Advent TurboROM, a MicroSphere RAM disk and printer buffer, ZCPR34, and a set of 9 extra keys. One of the keys does triple duty, since its return value, like that of many of the standard ASCII keys, is sensitive to the shift and control keys. I have configured most of the added keys as "function keys" which do system jobs rather than return characters. This article will describe how I use the ZCPR3 Input Output Package (IOP) to implement these functions. This involves a close and somewhat unorthodox relationship between the BIOS and the IOP. I'll cover each of these software systems separately in detail.

The End Result

This is the way my extra keys work. I have several pure function keys, called "blue keys" in the code, since that's the color of the key caps I salvaged from an old Kaypro II keyboard.

Blue key 1, above the ESCape key, performs a warm boot.

Blue key 2, just to the right of Blue Key 1, performs a cold boot.

Neither of these are maskable by software as long as the keyboard is beingchecked for input. They function similarly to Ctrl-Alt-Del on an MS-DOSmachine.

Blue key 3, to the right of the right cursor control arrow, toggles on andoff the high bit of ASCII characters typed at the keyboard, enabling thekeying of any 8 bit value.

Blue key "-", just below the Ctrl key, is rather like the Sys-Req key on theold PC's. It executes an RST 10h. Whatever code is placed at address 10hwill execute when this key is pressed. The default is initialized on a coldboot to a RET.

Blue key ".", to the right of the backspace key, can be shifted with boththe Ctrl and Shift keys. By itself, it toggles a pause in output from theMicroSphere printer buffer. Shifted, it initiates a screen dump to theprinter. With the Ctrl key, it clears the print buffer, aborting any printjob in progress.

Additionally, there are 4 "grey keys" in a line along the top of my numberkeypad. These keys return multiple ASCII values, determined by my runtimekey configuration software (KPAD and KVEC) and are frequently used withinapplications for setting margins, running special macros, displayingapplication help, and similar tasks.

Lindsay Haisley makes his home on the banks of beautiful Sandy Creek in Leander, TX where he lives with his wife Cheryl, their two pet birds, Chewey and Baby, and seven or eight computers of various sizes shapes and vintages. He travels several times a year to various parts of the country where he teaches and performs contemporary folk music at clubs, concerts and festivals. He has gained a reputation over the years as an autoharp player of note and is currently a staff writer for The Autoharp Quarterly, a national magazine for autoharp enthusiasts. When at home, he is the proprietor of Further Music Productions, a small music recording studio.

Lindsay graduated from Earlham College in 1963 with a BA in physics and math, and as a result, he takes his computer hobby quite seriously.His revision of P2DOS for CP/M, called NovaDOS, has circulated internationally, and can be found, along with a number of his other public domain offerings, in major CP/M archives such as Royal Oak and Simtel20. He is the sysop of the Znode 77 RAS which is owned by the Kaypro Club of Austin.

The Hardware

A Brief discussion of hardware problems is in order for anyone who might take this article seriously enough to try adding keys to his or her Kaypro keyboard. The key switches may be hard to find these days. Members of Kaypro users groups frequently have old keyboards around from which they salvage key switches and caps. The switches are almost identical to the ones used on the old Apple 2e computers, except that the stems are longer and the caps shorter. Both switches are made by SMK. The Apple key switch part number is SMK 705 0084 (I don't know the Kaypro part number). The Apple key switches were considerably cheaper than Kaypro key switches back when they were readily available through service centers; however, that may have changed now. Cutting spaces for the extra keys in the keyboard chassis may prove daunting. The chassis is solid steel, and cutting it requires some metal-working skills. One quickly realizes why the old Kaypros had a reputation for being indestructible! Farm the job out to a metal shop if necessary.

The BIOS

If the BIOS alone is programmed to perform the jobs to which my extra keys are dedicated, then the programming job required is relatively simple. Each time a keyboard char-

Listing 1

```
; CUSTOM BIOS CONST AND CONIN ROUTINES TO SUPPORT KAYPRO
; FUNCTION KEYS
;

; These are TurboROM entry points, defined in my BIOS.
;
        xentry  xcrtsta     ; standard keyboard input status
        xentry  xcrtin      ; standard keyboard input

maxkey  equ     92h         ; Largest return+1 from 'regular'
                            ; keys. Values above 7fh are
                            ; returned by the TurboROM for
                            ; the cursor control and number
                            ; pad keys for further processing.

; Tables for mapping keypad keys and cursor control keys
;
akeys:
    defb    0bh,0ah,08h,0ch ; Standard adm 3a cursor keys

keypad:
        defb    '0123456789-+'  ; Modified keypad definition
        defb    cr,'.'

;===============================================================
;       ** Console Status **

consta:
        .
        .  (IOBYTE routines, TurboROM motor
        .   timer sensing, etc.)
        .

bufclr  equ     $ + 1
        ld      a,0         ; Buffer clear flag
        or      a
        ld      l,0         ; Return null in L to indicate
                            ;  real char.
        ret     nz          ; Return if buffer not clear
                            ;  (bufclr set).
        ld      l,xcrtsta   ; Call the TurboROM keybd status
                            ;  to see if anything new.  A=0,
        call    calrom      ;  Zero flag set if not.
        ld      l,a         ; L must return zero if no char
                            ;  waiting.
        ret     z           ; Return to caller if no
character.
        ld      l,xcrtin    ; Else, fetch it with raw
                            ;  TurboROM conin.
        call    calrom

; Check for function keys pressed. Note that maxkey is set to
; include as 'normal' the TurboROM return value for the
; number pad and cursor control keys, even though they have
; their high bits set.

        cp      maxkey      ; Key >= maxkey=pure function key
        jr      c,isnorm    ; < maxkey = normal character or
                            ;  hybrid (extended input) key.

; If the key pressed is a function key, then we execute the
; following code.
;
isfunc: ld      l,a         ; Save char in L reg. for return
                            ;  to IOP
        xor     a           ; A=0 means say no key pressed.
        ret

; If the key is a regular ASCII key, then we proceed as
; follows. Since const cannot return the normal key byte,
; it is saved in bufchr (in conin) for the next conin call.
;
isnorm: ld      (bufchr),a ; An ASCII key has been pressed
        xor     a
        ld      l,a         ; Return null in L
        dec     a           ; Make A=0ffh

; We store 0ffh in bufclr (near the begining of consta) so
; that the next call to conin will access the character
; stored in ; bufchr rather than looking to the SIO for a
; key input.

        ld      (bufclr),a
        ret

;===============================================================
;       ** Console Input **

conin:
        call    consta      ; Check for key pressed.
        jr      z,conin     ; Loop out of ROM.
        ld      a,(iobyte)
;       etc.    ......
        .
        .

; If consta detected a valid character, then it placed it in
; bufchr.
;
bufchr  equ     $ + 1       ; One character look-ahead where
        ld      a,0         ;  const places characters.
        ld      hl,bufclr   ; Clear the character stored flag
        ld      (hl),0      ;  in const.

; The number keypad keys and cursor control keys are mapped
; by the TurboROM into values between 80h and 91h. Bit 7 of
; TurboROM's bsflag determines whether or not these values
; are further converted into the values provided in the akeys
; and keypad lists above.

mapfnc:
        or      a           ; Check for high bit set
        ret     p           ; If no high bit then return
        ld      hl,bsflag   ; See if we are to map
        bit     7,(hl)
        ret     nz          ; Return raw function if set
        ld      hl,akeys    ; Else, point to mapping table
        and     1fh         ; Strip high bits
        add     a,l
        ld      l,a         ; Get offset,
                            ;  (always in first page)
        ld      a,(hl)
        ret
```

acter is input from the keyboard SIO (serial input/output chip), the high bit is tested and appropriate action taken if it is set. If, on the other hand, we wish the BIOS to be transparent to these function keys—for processing to be done by a higher structure such as the ZCPR3 IOP—then the situation is not so simple. The BIOS has two routines which return information about the keyboard to the BDOS or other calling programs. At the lowest level, the BIOS CONST routine returns effectively a simple logical true/false value depending on the current availability of a key value for input from the SIO, while the CONIN routine "debriefs" the keyboard SIO, returning the key input and resetting the SIO so that subsequent calls to CONST return false until another key is pressed.

Because my function keys aren't true keys, but rather pushbutton switches which use the keyboard mechanism to communicate with the operating system, they really aren't entitled to return anything to calling software via CONIN. CONST must similarly return false (no key pressed) in response to a function key. How are we, then, to return a value from these keys using standard BIOS functions? The solution which I adopted relies on the fact that while CP/M requires the BIOS to return appropriate values in certain Z80 or 8080 registers, it assumes nothing about other registers, which may or may not have their contents preserved through a BIOS call. I have always considered this to be somewhat sloppy software design; however, it does allow my function key values to ride "piggyback" in an unused register, in this case the L register, on a call to the CONST routine.

Programs, including the BDOS, which call CONST expect nothing of significance to be returned in the L register and routinely discard it's contents. However, my IOP intercepts calls to the BIOS CONST and is written to interpret the contents of the L register and take the appropriate action. The situation is not a simple one. While a primitive CONST routine need only interrogate the keyboard SIO status register and determine the presence (or lack thereof) of a data byte, my CONST must additionally read this byte and, if it is not a function key byte, store it internally until requested to release it via a subsequent CONIN call. It must return a function key value in L once and only once (lest the requested function execute repeatedly). The CONST and CONIN routines must, therefore, work hand in hand to insure that they present a unified front to higher level software.

The BIOS console status and console input routines can be thought of as layered structures, paired to each other at each level. The introduction of an IOP adds yet another layer on which the relationship must be harmonious. At each level (although CONST may be called independently) the CONIN routine calls the CONST routine and loops until a character is available. The lowest level, on my system, lies within the

See IOP, page 55

Listing 2
```
;*************************************************************
     title   Custom IOP Kaypro 10, adds extra keys  10/13/89
     subttl  Documentation

; Package: NZIOP.Z80
; Author:  Joe Wright
; Date:    30 July 1987
; Version: 1.0

; Modified by Lindsay Haisley to support extra Kaypro keys

; This IOP implements several special function keys on the
; Kaypro 10 with an Advent TurboROM and enables the cursor
; control and number keypad keys to return strings of up to
; 4 characters each.

     eject
     subttl Mapping of special keys added to Kaypro Keyboard

; Blue Key Mapping
; ================
;
key1    equ     0feh        ; Blue Key marked "1"
key2    equ     0f0h        ; Blue Key marked "2"
key3    equ     0f5h        ; Blue Key marked "3"
dashkey equ     0b0h        ; Blue Key marked "-"
dotkey  equ     0e0h        ; Blue Key marked "."
shftdot equ     0eah        ; Shft Blue Key marked "."
ctrldot equ     0ebh        ; Ctrl Blue Key marked "."

; Now assign functions to blue keys
;
exwb    equ     key1        ; Execute warm boot
excb    equ     key2        ; Execute cold boot
pauspb  equ     dotkey      ; Pause printer buffer
haltpb  equ     ctrldot     ; Halt printer buffer
crtdmp  equ     shftdot     ; Dump screen to printer
sethbt  equ     key3        ; Pass next char with high bit set
rst10   equ     dashkey     ; Execute routine at 10h

       eject
       subttl  Equates and Macros

; ===========================================================
;
sndtab  equ     0febbh
sstat   equ     0fh         ; Status port, unused sio
iobyte  equ     0003h
cdrive  equ     0004h
bdos    equ     0005h
rom     equ     0fffch

        .
        .
        .
        . (ROM entry point generation, etc.)
        .
        . (Standard jump table, a la Joe Wright's
        .   NZBIO, v 1.0)
        . (IOP id's, etc.)
        .

; The main body of the IO Package starts here.
; The preceding jumps and package ID MUST remain in their
; present positions.  Code that follows is free-form.
;
status:                     ; Internal status routine
namer:                      ; Device name routine
newio:                      ; New i/o driver installation
                            ;  routine
copen:                      ; Open con: disk file
cclose:                     ; Close con: disk file
lopen:                      ; Open lst: disk file
lclose:                     ; Close lst: disk file
;
zero:   xor     a           ; Any call to this pack
                            ;  returns zero
        ret
        .
        .
        .
        . (Standard select routine)
        . (Standard tinit routine)
        .

; Tinit must set bit 7 of bsflag, the TurboROM configuration
; flag. Likewise, select, which is used to remove the IOP,
```

```
; must reset this bit. Bsflag bit 7 determines whether
; cursor control and number keypad keys are mapped in the
; BIOS or whether mapping responsibility is passed on to the
; IOP. I use a special entry in the BIOS jump table to return
; a pointer to bsflag. This BIOS vector (calspc) is
; referenced by a vector in the IOP, initialized by tinit.



; ===============================================
; Console Status
; ===============================================

nconst: ld      a,(cnt)     ; Get count of chars pending
        or      a           ;  from an extended key.
        jr      nz,loadup   ; If > 0, then return A=0ffh
        ld      a,(iobyte)
        rrca                ; Test for crt/tty
        ld      l,xttysta   ; Call ROM directly
        jr      nc,calrom   ; No buffering from tty

; Retchr will contain the return value from one of the grey
;  extra number keypad keys, if one has been pressed.

        ld      a,(retchr) ; Is a character waiting in conin?
        or      a
        jr      nz,loadup  ; If so, then return status true
;
        call    const      ; Call BIOS console status.
        ld      c,a        ; Put return value in C

; Since register L is used to return our "hidden" function
; code from the keyboard, we must check for reg L > 0.

        or      l          ; If both A and L zero,
                           ;  no key pressed
        ret     z          ; No L, just return A=0
        ld      a,c
        or      a          ; A and not L.  Regular key
        jr      nz,loadup  ; So return 0ffh

; This leaves only the option of a blue or grey key having
; been pressed, so .....

        ld      a,l        ; Get special key code

;
; Find the key code and jump to the appropriate routine
;
        cp      exwb       ; Execute warm boot
        jp      z,0000
;
        cp      excb       ; Execute cold boot
        ld      l,0
        jp      z,calrom   ; Use TurboROM for cold boot
;
        cp      pauspb     ; Pause print buffer
        ld      c,0
        jp      z,calspc   ; Special BIOS jump
;
        cp      haltpb     ; Halt print buffer
        ld      c,1
        jp      z,calspc   ; Special BIOS jump
;
        cp      crtdmp     ; Screen dump
        jr      z,dcrtdmp
;
        cp      sethbt     ; Set high bit
        jr      z,dsethbt
;
        cp      rst10      ; Execute RST 10
        jr      z,drst10

; Only keys left are grey keys (extra number pad extra keys.)

        ld      (retchr),a ; See note in IOP conin

; Return A=0ffh indicating a valid character waiting
;
loadup: xor     a          ; Valid character
        dec     a
        ret

; Toggle the setting of the high bit on all ASCII characters
; returned
;
dsethbt:
        ld      hl,himask  ; Call it from its lair
        ld      a,(hl)
        xor     80h        ; Amend it
        ld      (hl),a     ; And send it home
        xor     a
        ret

; The himask is set to 80h by the key which toggles the
; return of high bit set ASCII characters.
;
outkey:
himask  equ     $+1        ; Set msb
        or      00         ;  if requested to do so
        ret

; Execute the routine located at address 10h
;
drst10: rst     10h
        xor     a
        ret

; Screen dump function, a la TurboROM
;
;
dcrtdmp:
        ld      c,0ffh
loop1:  ld      l,xgetscr  ; Screen dump ROM entry
        call    calrom
        push    af         ; Save character
        ld      a,l        ; Test for end of line
        and     07fh       ; Mask column
        jr      nz,not0    ; Jump if not column zero
        ld      c,0dh      ; Start new line
        call    list       ; List is external to this listing
        ld      c,0ah
        call    list
not0:   pop     af         ; Get character
        jr      z,endscr   ; Dump ends with zero return
        ld      c,' '      ; Skip greek/foreign
                           ;  characters
        cp      c
        jr      c,grph
        ld      c,a        ; Get character to print
grph:   call    list       ; Send it to the printer
        jr      loop1
endscr: xor     a          ; Return with status false
        ret

; Special jumps added to my BIOS.  Calspc controls the
; MicroSphere printer buffer and returns the value of the
; TurboROM config byte. Calrom is a direct call to the
; TurboROM
;
calspc: jp      0          ; Filled in by tinit
calrom: jp      0          ; Filled in by tinit



; ===============================================
; Console Input
; ===============================================

nconin: call    nconst
        jr      z,nconin

; The cnt location contains the count of characters pending
; from an extended input key (number pad, cursor control or
; grey keys)
;
cnt     equ     $+1
        ld      a,0        ; Get counter value
        or      a          ; Any chars left from an
                           ;  extended key?
```

PMATE/ZMATE Macros

3. PMATE Facilities and Buffer-Saving Macros

By Clif Kinne

Although I did not consider features of PMATE, other than macros, to be within the scope of this column, the implementation and choice of macros is so intertwined with certain other facilities that that I feel I must discuss my use of them somewhat so that the basis for the choice and development of further macros will not be completely abstract, but be seen in the context of a specific framework.

PMATE Facilities

In my mind the facilities of PMATE relevant to macro considerations encompass:

Variables
Function Keys (if your system has dedicated f keys).
Buffers
Permanent Macro Area

Variables (numeric)

I pretty much covered my thoughts on variables in the last issue. To summarize those thoughts more explicitly:

1. The .^S (SaveEnv) subroutine macro pushes five variables and two cursor coordinates on the stack. To avoid frequent stack overflow, this is less than half of the 16 available stack levels.
2. Thus, these 5 variables, 0,1,2,7, & 8 in our case, should be preferred for data bytes which have to be retained while other macros are called.
3. Using V7 to save the home buffer when necessary, is an excellent example of such use. (Unfortunately, after I wrote the last column, I learned that Jay Sage is using V9 for that purpose. Had I checked earlier, I could have avoided that difference.)
4. The .C (Change) and .^P (Prompt) macros, introduced in Column #2, illustrated a common use for V8: the return of a single data byte from a subroutine to its calling macro.

Function Keys

If your keyboard has dedicated function keys, your decisions on how to use them will impact critically on the efficiency of your editing. Ten function keys allow you to invoke ten different macros with single, one-finger keystrokes. They should, then, be bound to macros you expect to use most frequently. This line of reasoning has led me to the following choices:

f1 Displays a directory of all editable files, with a "micro menu" on the command line.
f2 Toggles between Tag and Cursor
f3 Saves to disk the file in the current buffer. (See Buffer-Saving Macros below.)
f4 Toggles between the T-buffer and Buffer 1.
f5 Toggles circularly around Buffers 5-4-3-2-5-4-...
f6 Toggles circularly around Buffers 6-7-8-9-6-7-...
f7-f9 Invoke buffers 7-9 as macros.
f10 Clears the current buffer.

Buffers

T This, of course, is the primary buffer for file editing, being the only one capable of automatic disk buffering and generation of back-up files.
1 This is my second choice for text. The f4 key makes this very handy for such things as file comparisons, merging, et cetera.
2,3 These are sort of scratch buffers for mostly short- term jobs,- holding search- and replace-strings, for example.
4 This is the preferred choice for reference material, such as help files and others that can readily be recovered from disk if corrupted inadvertently.
5,6 These are my third choices for text. They can be alternated by pressing the two adjacent function keys, f5 and f6.
7-9 These, being executable by f7-f9, are left for that purpose unless no other buffer is free. (Jay has also, coincidentally, allocated these same buffers for this purpose.)

Since any contents of this buffer are wiped out by the next ^T, ^E block command, I try to use it only with macros that do their jobs and terminate. I believe that if you set up your own guidelines, such as these, you will find it easier to write macros that minimize the chance of wiping out wanted text (or, alternatively, aborting macros to avoid it).

Permanent Macro Area (PMA)

This facility deserves a column of its own, including some macros for dealing with it. In addition, you will have noted that I have alluded to other macros and matters that deserve more attention, and which could be deferred

Clif Kinne is a retired computer designer. He cut his teeth on vacuum tube and acoustic delay line machines in the fifties, made the transition to transistors and magnetic cores in the sixties, left the field to his children in the seventies, and tried, vainly, to catch back up with them in the eighties. He can be reached by voice at 617-444-9055, or via a message on Jay's BBS, 617-965-7259. His address is 159 Dedham Ave., Needham, MA 02192

Listing 1. Revisions of .^S and .^R macros.

```
^X^S    ;SaveEnv    Rev A.                                  42 bytes

;       FUNCTIONAL SPECIFICATION:  Pushes variables 0, 1,
            2, 7, 8, and cursor position on the stack.
;           Loads variable, V7, with current buffer, @B.
;           IF 0 argument, prepares B0 to receive string.

;       STACK USE:      7 out of the 16 available levels.

;       USAGE:    .^S    Leaves Buffer 0 alone.
;                 0.^S   Prepares B0  to receive string.
;           Normally called at the start of any macro which
;           will alter more than one of the items saved.

        ; Original code:
@0,@1,@2,@7,@8, ;Push variables V0,V1,V2,V7,V8 on stack.  1
@X,@L,          ;Push current col. and line no. on stack. 2
@BV7            ;Put current buffer no. in variable, V7.  3

        ; Rev A additions:
@0%             ;IF argument was not 0, terminate.        4
BEA             ;ELSE go to top of Buffer 0.              5
27I             ;Insert an ESCAPE (use "6/2I for radix    6
                ;  invariance).
"%I             ;Insert a % sign.                         7
A               ;Return to top of Buffer 0.               8
B@7E            ;Return to home buffer (ZMATE & PCMATE)   9
;@7.^G          ; Alternate line 9 for MATE
```

```
^X^R    ;Restore    Rev A.                                  61 bytes

;       FUNCTIONAL SPECIFICATION:  Restores the environment
            saved by the SaveEnv macro.

è;       USAGE:  Must be invoked when exiting a macro which
;               has used SaveEnv, .^S, whether that exit
;               was normal, a conditional exit, a jump, or
;               a programmed abort.
;               If the macro is aborted with  Ctrl-C or
;               terminated accidentally, the variables must
;               be reinitialized, manually or otherwise.

        ; Rev A additions:
BEA             ;Go to top of Buffer 0                    1
E               ;Disable error messages                   2
1S^L$$          ;Search for an ESCAPE on topline          3
@EJO            ;IF not found, jump to origninal code.    4
@T="%           ;IF found and next character is %,        5
  {D            ;  THEN delete it and                     6
   0K           ;    delete line left.                    7
  }             ;END IF found                             8

:O      ; Original code:                                  9
B@7E            ;Return to home buffer (ZMATE & PCMATE)   10
;@7.^G          ; Alternate line 7 for MATE
@S-@LL          ;Move from cur. line (@L) to saved line.  11
QR              ;Refresh screen (avoids format mode bug)  12
@SQX            ;Move to saved column no. (@S).           13
@SV8 @SV7       ;Restore original contents to V8 & V7.    14
@SV2 @SV1 @SV0  ;Restore original contents to V2,V1,& V0. 15
```

Listing 2. Macro to save non-open files.

```
^XO     ;SavBuf                                         154 bytes

;       FUNCTIONAL SPECIFICATION:

;           1.  IF fn (filename) is not found in the top
;               line, requests fn from user.
;           2.  Copies fn to top of Buffer 0.
;           3.  Checks if file is on disk
;               IF so, asks permission to delete it.
;           4.  IF permission given, deletes (XX) old file.
;           5.  Outputs (XO) the buffer contents to disk.
;           6.  Restores environment and terminates.
;
;       VARIABLES USED: V8  Set by .^P if user types an ESC

;       BUFFERS USED:   0       Holds filename

;       SUBROUTINES:    .^C     Confirm.    USING: .^Y
;                       .^P     Prompt.
;                       .^R     Restore.           .^G
;                       .^S     SaveEnv.           .^G
;                       .^Y     Yes                .^A
è
;       SIDE EFFECTS:  As written, this will not preserve
;               the tag or the cursor position in Buffer 0,
;               nor the tag in the home buffer.

;       USAGE:  This may be invoked from any buffer except
;               B0.

0.^S            ;Save the environment. (Note 0 argument)  1
A               ;To top of buffer.                        2

[           ;Find first non-space, non-ctrl character     3
  @T>32_        ;IF character > ASCII space, escape loop  4
  M             ;ELSE move to next character.             5
]               ;                                         6
(@T="/)         ;IF first of a 2-char. ('/*') comment     7
  {M}           ;  lead-in, move to next character.       8

;Is it a comment lead-in?
        ;IF a comment
(@T=";          ;for PMATE or Assy language               9
  !(@T="{)      ;  OR Pascal                              10
  !(@T="*)      ;  OR C (2nd character)                   11
)&              ;AND                                      12
  (@L=0)        ;  on the top line,                       13

{           ;THEN copy fn into Buffer 0:                  14
  M             ;  Move off the lead-in character.        15
  T             ;  Tag first letter of fn.                16
  2[            ;  Start loop. (2 is for fn + ext.)       17
    M           ;  Move to 2nd letter                     18
    S^W$        ;    Find 1st non-letter/non-digit.       19
    -M          ;    Move back onto character.            20
    @T='.'      ;    IF not a period, escape loop         21
  ]             ;    ELSE loop to get extension.          22
  #BD           ;  Duplicate the fn in buffer 0.          23

}{          ;ELSE (no embedded fn) get fn from user:      24
  :R            ;  Label to Reask for filename.           25
  .^PFilename:$ ;  Display prompt and get user's response 26
  #BN           ;  Move the fn typed to Buffer 0.         27
  -L3K          ;  Remove the prompt space.               28
  @8            ;  IF abort flag is set,                  29
    {GAbort save?$;   Verify                              30
    .^Y@S       ;     IF 'yes'                            31
      JT        ;       jump to Terminate                 32
    JR}         ;     ELSE reask for filename.            33
}           ;END of IF THEN ELSE                          34

;Filename is now in Buffer 0:

è@F^A@0$        ;Is a file of that name on disk?          35
{           ;IF so,                                       36
  .^C           ;  Ask if okay to delete.                 37
  @S'           ;  IF not okay,                           38
    JT          ;    Terminate gracefully.                39
  XX^A@0$       ;  ELSE delete existing file from disk.   40
}           ;END IF so.                                   41

XO^A@0$         ;Output the Buffer contents to disk.      42
:T              ;Terminate                                43
.^R             ;Restore the environment.                 44
```

over a year or more if I am left to my own devices. If any of you would like to hear more about a particular macro, or other item, a word from you could greatly alter my schedule.

Buffer-Saving Macros
(or, efficient use of the XE, XJ, XO, and XH commands)

Here we shall cover three macros:

.O (SavBuf) A substitute for the XO command, bound to the ALT O instant command in my PCMATE. (For MATE and ZMATE options, see the section, "Key Bindings", in Jay Sage's Z-System column in *TCJ* issue 46.)
.3 (SavFil) My universal Save command, bound to f3.
.Q (QuitMate) A macro to terminate PMATE, bound to ALT Q in my PCMATE.

These are among my most important and frequently used macros and illustrate, I think, why I felt impelled to include the above discussion, especially on function keys. Before introducing them, I must make two preparations.

Embedding a filename in a file

First let me describe a technique for having a text file carry its own name. This was introduced in the 10/83 *Lifelines* by Todd Katz and has saved me hours of typing and who knows how many mistakes.

Right here I don't think I can improve on Katz so I shall quote: "The trick is to put the name of the file as a header line preceded by a semicolon" (or other comment delimiter if it is a source file for another language). Of course, after a space or a tab you could put the date or other comments on the same line.

You can use this technique to name other text files as well as source code. To avoid having the header line print with XT, simply precede the comment delimiter with a Ctrl F.

Enhancements of the SaveEnv and Restore subroutines.

The second preparation is effected through revisions to two of the macros introduced in column 2.

Our principal macro, SavBuf, has to store a string (the filename) in a buffer for subsequent reference. Certainly we shall want to preserve that buffer's contents, since we shan't want anything to deter us from hitting the SaveFile key when we think of it. We can leave the contents of that buffer undisturbed, so long as we put the string at the top and terminate it with an ESCAPE.

Recall that we did a similar thing in column 1 with the macro, .D, except that that time buffer 0 was to be called as a macro, so the string terminated with a %, rather than an ESC. These two kinds of uses for buffers occur rather frequently. If we use Buffer 0 for both, we can expand .^S to prepare Buffer 0 for these uses and .^R to restore it afterwards. *Listing 1* presents revisions of .^S and .^R to achieve this.

For .^S, note that:
1. Unless it is called as 0.^S, it terminates before executing the new code.
2. Otherwise it inserts an ESCAPE followed by a % at the top of B0, in preparation for a string to be inserted before them by the calling macro.
3. Such a string will then be recognized as ^A@0, or be executed as a macro by .0 without disturbing, or being affected by, other contents of the buffer.

For .^R, note that:
1. It searches for an ESC,% sequence on the first line and if found deletes it and any preceding string.
2. It is possible that ESC,% could be on the top line of B0 without having been put there by .^S, but I'm willing to risk it. If someone sees a simple way of removing all doubt, I shall be glad to pass it on in a subsequent column,- with credit, of course!

The SavBuf macro, .O (See Listing 2).

Some comments are in order to amplify those in the Listing. (I try to strike a balance between having enough comments that the listings stand alone, yet few enough that I don't divert attention from the main train of thought).

1. The listing illustrates code that will recognize both 1-character and 2-character comment delimiters, as used with Pascal and C source, code as well as the ';' for Assembly and PMATE. You should include delimiters for the languages you use.
2. The code checks that the filename, if found, is on the top line of the buffer. I see no reason why it has to be there, but neither do I see any reason not to enforce that much uniformity.
3. If the filename is not embedded in the file, the Prompt macro, .^P, is invoked to get it from the user. Since this gives the user a chance to change his mind and abort, .O recognizes the abort flag and responds appropriately.
4. Though I most frequently use this to save files in other than the T Buffer, it is often convenient to use it for saving a file, open in the T buffer, to another name.

The SavFil macro, .3 (See Listing 3).

This listing is pretty self sufficient. Note the use of @N. This is a flag that is set if there is a file open. I believe it is only available in PCMATE versions. For others I have offered substitute code that queries the user. If you find this interruption annoying (as I do), you may prefer to forbid yourself to use the T buffer for unopened files, and omit the code on lines 2 (2a & 2b) and the close- brace on line 9. Then, if you embed filenames, you can hit your equivalent of my f3, anytime you want without a thought as to which buffer you're in or what file is in it.

The QuitMate macro, .Q (See Listing 4).

This macro tries to make it easy for you to leave PMATE without leaving changes unsaved. It asks if you want to save the buffer you're in when you call it. If, in addition, there is something in the T-Buffer, it asks if you want to save that also. Of course, there might be a change to a third or fourth buffer, but it is hoped that these two queries will jog your memory enough that you won't forget them.

The same remarks on the use of @N apply as in the foregoing SavFil macro, but to lines 18, 18a & 18b instead of 2, 2a &2b.

Macros covered in columns 1, 2 and 3.

This seems like a good time to refresh our memories on macros presented so far. Table 1 does this, and I shall try to include an update in each issue, when space permits.●

Listing 3. Universal Save File Macro.

```
^X3     ;SavFil                                          20 bytes

;       FUNCTIONAL SPECIFICATION:

;       1.  IF in T-buffer, checks if file is open.
;             IF so, saves file to disk using XJ.
;       2.  ELSE saves by calling .O.
;       3.  In either case leaves cursor position as it was.

;       SUBROUTINES:            USING:
;               .^S     SaveEnv.
;               .^R     Restore.
;               .O      SavBuf.         V8, B0, .^C, .^P

;       USAGE:  This should be bound to an easily typed
;               instant command, preferably a dedicated
;               function key.  Can be called from any
;               buffer except B0.

@B=0            ;IF in T-buffer,                                 1
  {@N           ;  THEN IF file is open,                         2
;       If you don't have @N, substitute lines 2a and 2b:
; {GIs this file open?$.        ;Query user.                     2a
; ^Y@S          ;  THEN IF answer is 'Yes',                      2b
    {           ;    THEN                                        3
    .^S         ;    Save the env. (cursor location).            4
    XJ$         ;    Save file to disk and reopen.               5
    .^R         ;    Restore the env. (cursor).                  6
    %           ;    Terminate the macro.                        7
    }           ;  END IF open                                   8
  }             ;END IF in T-buffer                              9
                ;ELSE (unopened file in any buffer):
.O              ;Call SavBuf macro.                              10
```

Listing 4. Macro to exit PMATE.

```
^XQ     ;QuitMate                                        88 bytes
è
;       FUNCTIONAL SPECIFICATION:

;       1.  If buffer not empty, displays query, "Save?".
;       2.  Any response but 'Y' or 'N' aborts macro.
;       3.  For "Yes", saves current buffer (XE or .O)
;       4.  If current buffer not T, moves to T-buffer and
;           repeats steps 1. and 2.
;       5.  Exits PMATE (XH).

;       SUBROUTINES:            USING:
;               .^N     No              .^A
;               .^Y     Yes             .^A
;               .O      SavBuf.         V8, B0, .^C, .^P

;       USAGE:  This is best bound to an instant command.
;               Can be called from any buffer except B0.


:A              ;Restart label                                   1
@C!@T=0{        ;IF current buffer empty, THEN                   2
  @B{           ;  IF not in T-buffer THEN                       3
    BTE         ;    go there                                    4
    JA          ;    and restart.                                5
    }{          ;  ELSE (in T-buffer)                           6
    XH          ;    Leave PMATE.                                7
    }           ;  END IF THEN ELSE.                             8
  }             ;END IF buffer empty                             9

        ;Current Buffer is not empty:
QB              ;Prompt alert.                                   10
GSave?$.^Y      ;  Save current buffer before quitting?          11
@S{             ;  IF 'Yes' THEN                                 12
  @B{           ;    IF not in T-buffer, THEN                    13
    .O$         ;      Call routine to save non-open file        14
    BTE         ;    Move to T-buffer                            15
    JA          ;      Restart                                   16
  }{            ;  ELSE (in T-buffer)                           17
    @N{         ;    IF an open file, THEN                       18
;       If your PMATE does not support @N, use 18a & 18b
;   GIs this file open?                                          18a
;   .^Y@S{      ;      IF 'Yes', THEN                            18b
      XE$       ;        Close it and save to disk.              19
    }{          ;      ELSE (file not open)                      20
      .O$       ;        Save non-open file.                     21
    }           ;    END IF THEN ELSE open                       22
    XH          ;  Leave PMATE                                   23
  }             ;  END IF THEN ELSE not in T-buffer              24
}{              ;ELSE  (not 'Yes' to 'Save?')                    25
  .^N@S'        ;  IF not 'No',                                  26
    %           ;    abort macro                                 27
  @B            ;  ELSE ('No') IF not in T-buffer                28
è {             ;  THEN                                          29
    BTE         ;    Move to T-buffer                            30
    JA          ;    and restart.                                31
  }             ;  ELSE (in T-Buffer)                            32
    XK          ;    Close file (if open)                        33
    XH          ;    and leave PMATE.                            34
}               ;END IF THEN ELSE 'Yes' (to 'Save?')             35
```

Table 1. MACROS LISTED TO DATE.

My ID	My Name	TCJ No.	My ID	My Name	TCJ No.
.^A	Answer	49	.^R	Restore	49,50
.^B	BufferTest	49	.^S	SaveEnv	49,50
.^C	Confirm	49	.^Y	Yes	49
.^D	DigiTest	48	.3	SavFil	50
.^G	GoBack	49	.C	Change	49
.^N	No	49	.D	Decimal	48
.^P	Prompt	49	.O	SavBuf (XO)	50
.^Q	Quit	49	.Q	QuitMate	50

Corrections: Please add the following lines to the ends of listings 1, 2 and 3 from the previous issue. They were inadvertently dropped during typesetting. My apologies to Clif and to you—Editor.

In *Listing 1*, after the lines reading:

```
;       @C!@T = 0 only if both @C and @T are 0, which is
;                       TRUE only for an empty buffer.
```

Add the following:

```
@C!@T=0 ;IF Buffer is empty,                                     1
{       ;THEN                                                    2
  -1,   ;  Push TRUE on the stack                                3
}{      ;ELSE                                                    4
  .^C   ;  Ask if ok to delete buffer. Push answer               5
}       ;  on stack (TRUE if "Y", otherwise FALSE).              6

;Compact form:  @C!@T=0{-1,}{.^C}
```

In *Listing 2*, after the lines reading:

```
@S-@LL          ;Move from cur. line (@L) to saved line.         2
@SQX            ;Move to saved column no. (@S).                  3
```

Add the following:

```
@SV8 @SV7       ;Restore original contents to V8 & V7.           4
@SV2 @SV1 @SV0  ;Restore original contents to V2,V1,& V0.        5
```

And in *Listing 3*, after the lines reading:

```
  @K=127{       ;  IF a DELETE (127),                            12
    -D^}        ;    delete previous character and loop.         13
```

Add the following:

```
  @XI           ;  ELSE display character on CRT & loop.         14
]               ;End of loop.                                    15
```

Z-Best Software

We're Off to the Libraries

By Bill Tishey

First, a quick thank-you to Bruce Morgen and Richard Swift who took time to report some typos and mixups in several of my Z3HELP files and ZFILEVxx.LST. Such cooperation is greatly appreciated. These two works represent an enormous amount of information and I'm sure still contain some misprints or even misinformation. Please continue to pass along any errors that you may find, so that we can make them as accurate as possible. I'll discuss the Z3HELP system in depth next issue.

The Libraries (SYSLIB, VLIB, Z3LIB, DSLIB, ZSLIB) (ZSUS Z3PROG Pack)

There has been a great deal of confusion over the past few years about the status of the Libraries, particularly as many programmers are used to finding the source for ZCPR in the public domain. Richard Conn released all of ZCPR30 to the PD, including SYSLIB (3.6), VLIB and Z3LIB. Source for the newer versions of the Libraries, however, under development by Joe Wright and Hal Bower, has *not* been released. This has been necessary simply for the purposes of edition control. Joe and Hal have plans for still other improvements and, for a while at least, as they approach the end of Version "4", they will continue to release only the relevant .REL and .HLP files for public use. At the end of their effort, they intend to offer a Reference Manual and source code at a reasonable price. So, although the source will not be "free" as before (at least for some time), we can rest assured that some means will be devised to make it available at reasonable cost to those who have a need for it. If you program for Z-System, there are many in the Z community (if not Hal and Joe themselves) who can work with you in the interim on any difficulties you may with the routines. Hal and Joe, I'm sure, would also like to hear of any problems which surface with the Libraries and any suggestions you have to improve them. The *Reader-to-Reader* column here might be a good forum for such discussions.

One problem which Hal Bower has mentioned in reference to public access to the source code to the Libraries is the failure of some programmers to anticipate problems which can result in the way they use certain routines. An example is some of the liberties taken with the system registers which may wreak havoc on some systems where these have an essential use. Programming for Z-System, in particular, must be done in a highly *defensive* manner. Routines must not only be used for the purposes for which they were intended, but also with considerable thought to their affect (i.e., compatibility) in a wide range of uses.

Name	Vers	S	ZSUS	Siz	Rec	CRC	Library/Size	Issued	Author
DSLIB .REL	4.3a	4	PROG	6	44	E090	DSLIB43A	24 01/01/91	Harold Bower
DSLIBS .REL	4.3a	4	PROG	6	41	68B0	DSLIB43A	24 01/01/91	Harold Bower
SYSLIB0 .REL	4.3c	4	PROG	13	103	937E	SLIB43C	45 12/12/90	Harold Bower
SYSLIBS0.REL	4.3c	4	PROG	12	93	F6C2	SLIB43C	45 12/12/90	Harold Bower
SYSLIB1 .REL	4.3c	4	PROG	8	63	6CC3	SLIB43C	45 12/12/90	Harold Bower
SYSLIBS1.REL	4.3c	4	PROG	8	61	65C7	SLIB43C	45 12/12/90	Harold Bower
VLIB .REL	4.3	4	PROG	6	43	4D3C	LIBS43	45 08/18/90	Harold Bower
Z3LIB .REL	4.3b	4	PROG	11	87	9A12	Z3LIB43B	26 02/17/91	Harold Bower
Z3LIBS .REL	4.3b	4	PROG	10	79	33A9	Z3LIB43B	26 02/17/91	Harold Bower
ZSLIB .REL	2.10	4	V103	6	45	08F7	ZSLIB21	78 02/02/90	Carson Wilson
ZSLIBS .REL	2.10	4	V103	6	47	425D	ZSLIB21	78 02/02/90	Carson Wilson

Figure 1

One can view the Libraries in terms of progression from very generalized to more specialized routines. SYSLIB contains basic routines which access features of both vanilla CP/M and Z-System; Z3LIB adds general routines which provide access to ZCPR3 features and capabilities; VLIB adds still more specialized routines which support CRT control and utilization via the Z3TCAP; and DSLIB adds routines supporting file time and date stamping and real-time clock features. ZSLIB, developed by Carson Wilson, provides additional routines for datestamp maintenance under ZSDOS, Z3PLUS and CPM Plus.

Hal Bower has been hard at work in recent months with updates to SYSLIB, (4.4 should be available as you read

Bill Tishey has been a ZCPR user since 1985, when he found the right combination of ZCPR2 and Microsoft's Softcard CP/M for his three-year-old Apple II+. After graduating to ZCPR30 and PCPI's Applicard CP/M, he did a "manual install" of ZCPR3.3 (with help from a lot of friends!), and in late 1988 switched to NZCOM and ZSDOS, all on the same vintage Apple II+. Bill is the author of the Z3HELP system, a monthly-updated system of help files for Z-System programs, as well as comprehensive listings of available Z-System software. Bill is the editor of the Z-System Software Update Service and has compiled such offerings as the Z3COM package and the Z-System Programmer's Toolkit. Bill is a language analyst for the federal government and frequents the Foreign Language Forum (FLEFO) on Compuserve. He can be reached there (76320,22), on Genie (WATISHE), on Jay Sage's Z-Node #3 (617-965-7259) and by regular mail at 8335 Dubbs Drive, Severn, MD 21144.

this), Z3LIB (now 4.3b) and DSLIB (4.3a). See *Figure 1* for stats on the most recent releases (both Microsoft and SLR .REL formats are available for most of the libraries). With release 4.3 last August, Hal consolidated all the changes since 4.2 to make the version numbers the same and establish a more controlled baseline. He also added a module, LIBVERS.REL, which, when linked with SYSLIB/Z3LIB/VLIB/DSLIB and executed, displays version-specific data for each Library.

The major improvement to SYSLIB this past December (version 4.3c) was inclusion of Joe Wright's new, general-purpose sort routine. Now much faster and smaller, it replaces the unique, embedded sorts used in the directory routines and provides primitives for true relational database management for CP/M: sorted arrays, pointers to pointers, searching an array that has been sorted, et cetera.

One side-note to Hal's work with SYSLIB has been having to cope with its sheer size. The source to SYSLIB is now over a megabyte, which presents a problem to some library managers which must build the library entirely in memory (Al Hawley's ZMLIB, we should note, operates in a paged mode and doesn't have this limitation). As a result, at one point, SYSLIB was released in two pieces (SYSLIB0.REL and SYSLIB1.REL), but this proved to be too much of a nuisance (some assemblers and linkers don't handle 7-character labels) and, with version 4.4, we're now back to a single .REL file. If you happen to have the split-file version, however, just remember to rename the files (e.g., SLIB0.REL and SLIB1.REL) before linking. If you have the ZMAC package, you can concatenate the two .RELs with "ZMLIB SYSLIB = SYSLIB1, SYSLIB0".

DSLIB and Z3LIB have both needed tweaking to take advantage of the new SYSLIB improvements. The DSLIB directory routines (DDIRQ, DDIRQS), for example, were modified to return both the address of the sort table *and* the record table. This allows use of the new sort routine with pointers to perform customized directory lists with date and time stamps (by selecting/packing/ resorting, etc.).

One final note is a caution which Hal has made in the past to programmers using Z3LIB, and it relates to his emphasis on writing code in a defensive manner: "In many cases the Z33 functions in Z3LIB rely on the CPR to do their work. If a program overwrites the CPR, these routines probably will *not* work. Many programs (VLU, for example) may bomb because of this, either directly or indirectly."

Text File Extenders

EXTEND13 (ZSUS Vol 1 #4)

CONCAT13 (ZSUS Vol 2 #7)

Utilities to append strings and entire files to other files or to concatenate two or more files into one have been available for CP/M about as long as the original PIP.COM, which, although primarily a file copy utility, is also capable of the latter. Two other useful text file extenders, for vanilla CP/M, are Joe Wright's CON20 (good for appending a small file to the end of another file) and Ron Fowler's EXTEND (for appending ASCII strings to the end of a file from the command line). TTOOLS, in our Z-SUS Word-Processing Pack, is a set of UNIX-style utilities which also contains some useful "pipelines" for combining files under CP/M.

For both CP/M and Z-System (NZCOM, Z3PLUS), Carson Wilson's EXTEND13 stands out. Based on Ron Fowler's program, EXTEND13 appends an input line to a new or existing ASCII file (see *Figure 2* for EXTEND13's syntax). Great for batch- processing under Z-System, EXTEND13 can "keep notes" on what happened during unattended batch runs.

Figure 2

```
Name          Vers   S ZSUS Siz Rec CRC  Library/Size Issued   Author
============ ====== = ==== === === ==== ============ ======== ================
EXTEND.COM   1.30   4 V104   1   8 B1B2 EXTEND13  10 03/16/90 Carson Wilson
Text file extender for all Z80 machines.  Appends input line to new or
existing ASCII file.  Vs. 1.0 (09/81) by Ron Fowler.

Syntax:  EXTEND <filename> <string>    ]
```

Probably the best utility in this category to appear in the past few years, however, is Gene Pizzetta's CONCAT13. Like Gene's XFOR, which we discussed last issue, CONCAT is the result of several years of collaboration among Z-programmers and Z-users to produce a highly efficient and functional program. Howard Goldstein, Jay Sage (and undoubtedly others) assisted Gene with many programming decisions and many users helped to test new options and versions as they appeared.

For ZCPR3 only, CONCAT will either concatenate two or more source files into a new file or append them to an existing file. See *Figure 3* for CONCAT's syntax.

Figure 3

```
Name          Vers   S ZSUS Siz Rec CRC  Library/Size Issued   Author
============ ====== = ==== === === ==== ============ ======== ================
CONCAT.COM   1.30   4 V207   7  52 A98D CONCAT13  43 02/23/91 Gene Pizzetta
Concatenates two or more source files into a new file, similar to PIP or
appends them to an existing file. Accepts both DU and DIR specs. ZCPR3 only.

Syntax: CONCAT {dir:}outfile={dir:}infile {{dir:}infile {...}} {/options}    ]
```

Since CONCAT is Z-aware, it offers many special features. Since DU/DIR specs are allowed, some elaborate concatenations are possible across drives and user areas. In concatenate mode (the default), the file create datestamp is transferred to the new file (in append mode, the original create stamp is preserved). An object (binary) file mode is also available for "rejoining" files split on record boundaries (such as those created by FSPLIT, which allows splitting of binary files, or some hard-disk backup programs which break large files across two or more floppy disks). A divider string can be inserted into the destination file before it is concatenated or appended, and the system date and time can also be inserted (an included patch file even allows you to change the format of these to your liking). Other command-line toggles include: quiet mode, datestamp transfer, and disk-space checking. CONCAT is configurable with ZCNFG, so if your usage mostly entails appending files, you can make this the default mode as well as change other defaults.

FF24

(ZSUS Vol 2 #5)

The Z-System File Find program is one of those indispensable system utilities which performs a basic function, is used

See Z-Best, page 54

REAL COMPUTING

The 32FX16, CPU caches, and the Pi benchmark

By Richard Rodman

32FX16

The 32FX16 is an NS32 CPU designed for the explosive fax and fax-related marketplace. It is designed to make a fax-related product inexpensive, yet feature-rich. Like the 32CG16, it has an on-chip clock generator, enhanced graphics instructions, and can use an FPU, but not an MMU. It's packaged in an inexpensive 68-pin PLCC package.

Unlike the 32CG16, the 32FX16 has an on-chip "DSP module". The DSP module is intended to implement modems, tone decoders or other digital filtering applications. It performs one of four operations on a complex vector (that is, an array of complex numbers, each of which has a 16-bit real part and 16-bit imaginary part): VCMAD (Vector Complex Multiply Add), VCMUL (Vector Complex Multiply), VCMAC (Vector Complex Multiply and Accumulate), and VCMAG (Vector Complex Magnitude). These instructions run in parallel with whatever the FX16 CPU is doing. The DSP has 6 registers and is memory-mapped at address FFFFD400.

The 32FX16 also has 384 bytes of on-chip RAM, which can be used with the DSP or for any other purpose. It is located at FFFFD000.

A companion part to the 32FX16 is available, the 32FX210 facsimile/data modem analog front end. This device is a combination of a CODEC (coder/decoder, in other words, a combined A/D and D/A converter) and a filter device. The CODEC includes programmable gain and "mu-law" response, which is the standard North American companding method. While this is handy for telephony, to write a modem it will be necessary to convert the response back to linear values with a lookup table.

There is an evaluation/development board available, which is a PC/AT motherboard form factor board with four XT-compatible slots and from 2 to 8 megabytes of SIMM DRAM. For those itching to get into the fax machine business, National has available a complete software package including the complete V.29, V.27 and V.21 modems, the T.30 initial handshake, and the T.4 (Group III) compression/decompression logic. Plug in your scanner and printer, and an LCD display and keypad, and a DAA, and you're in business.

Needless to say, prices on this kind of thing are outside the experimenter's budget. But the 32FX16 has enough interesting aspects to it that, if low in cost, it could be useful for all kinds of neat projects.

Cache on the Barrelhead

In the marketspeak of the 32-bit CPUs, one word often heard is "cache". While National and Motorola have had on-chip caches for years, the word is new to the ears of the Intel-processor community. Intel has added a "four-way set-associative cache" to their 80486 CPU. The PC journalists have seized the opportunity to dredge up all of the old "cache" puns - after all, it's not a new concept by any means. But just what the heck is a "four-way set-associative cache", anyhow?

A cache is a block of high-speed memory which keeps copies of recently-accessed, and hopefully frequently-used, memory locations. It can be implemented on the CPU chip, or on external chips or boards. Whenever the CPU accesses a location, the cache logic checks to see if the location matches any of those in the cache. If it matches, the CPU saves time getting its data. You could call it a "cache discount."

The cache itself consists of the memory itself plus a list of addresses to which the cache memory locations are mapped, called the "tag directory". Matches are referred to as "cache hits", whereas the times when the CPU must go and access main memory are referred to as "cache misses". The goal of the cache design is to attempt to keep the ratio of hits to misses high, because the misses can be expensive. You could call them being "short of cache".

The locations in the cache are not assigned randomly. If the entire block of cache memory corresponds to a single block of main memory, this is called a "direct mapped" cache. Otherwise, the cache memory can be divided up into sections, each of which corresponds to a different block of main memory. This is referred to as an "N-way set-associative" cache, where N is the number of sections. This improves throughput, because the CPU is usually accessing more than one location in memory during a given set of instructions. It's like being in a store with multiple "cache registers."

The Intel 486 processor has an on-chip 8 kilobyte 4-way set associative cache. This means that its cache is divided into four sections, each of which could correspond to instructions or data. The cache has a dramatic effect on performance, as you will see in the "Pi benchmark" section to follow. You could call this "cache on delivery".

How does this newfangled 486's cache compare to the NS32 chip caches, available in the 32532 since 1986 and later in the 32GX32 and 32GX320? Internally, the '532 and 'GX32 processors have a RISC-like

See Real, page 49

Rick Rodman works and plays with computers because he sees that they are the world's greatest machine, appliance, canvas and plaything. He has programmed micros, minis and mainframes and loved them all. In his basement full of aluminum boxes, wire-wrap boards, cables running here and there, and a few recognizable computers, he is somewhere between Leonardo da Vinci and Dr. Frankenstein. Rick can be reached via Usenet at uunet!virtech!rickr or via 1200 bps modem at 703-330-9049.

Assembler, from page 30

ZREMOT03.LBR contains five modules that are assembled and linked to produce the final ZREMOTE program for Cam Cotrill's Ampro system.

```
ZRLDR.Z80       ZR program signon, etc. Calls RSXLDR.
RSXLDR.Z80      Generic RSX installation code. Calls RELOC.
RELOC.Z80       Generic code relocator for PRL modules.
ZREMOTE.Z80     Main ZREMOTE RSX hardware independent code.
ZRAMPRO.Z80     Hardware specific part of the RSX code.
```

The only module that changes for other hardware is ZRAMPRO, the hardware-dependent driver. If you have an IMP or MEX hardware insert (also called overlay) for your system, you can get the correct port addresses and bit mask data from it for modification of ZRAMPRO. After making the modification for my S100 system, I renamed this module to 8250IO.Z80. Lets assume you name your ZRAMPRO replacement MYDRVR.

Assemble all the modules to make REL files. (MYDRVR instead of ZRAMPRO)

The next step is to combine all these into a file named ZREMOTE.COM whose code is arranged like this, where all parts are present:

```
[RSX_HDR] [PRL_HDR] (CODE_IMAGE) [BIT_MAP_if_PRL]
```

We will use ZREMOTE.HDR as the name for RSX_HDR. This file is the same as a .COM file and must be made with a simple linking operation.

ZREMOTE.HDR is made with:

```
ZML ZREMOTE.HDR=ZRLDR,RSXLDR,RELOC
```

The next task is to link ZREMOTE.REL with MYDRVR.REL as a PRL, prefixed with ZREMOTE.HDR to make ZREMOTE.COM:

```
;                           LISTING 1.
;Program SETNULL
;Author Al Hawley
;Version 0.1, 3/18/91

;This program silently sets the NULLS variable in BYE
;or ZREMOTE to values from 0 to 254. Because of the
;nature of the routine in ZREMOTE/BYE which handles this
;task, the nulls cannot be set to 255.

;       .Z80            ;uncomment this line for M80 only

include         sysdef          ;standard equates
.request        syslib          ;tell linker about syslib
ext             eval,sksp,print ; and the routines we use
; from syslib

;-----
;       START OF PROGRAM CODE
;-----

NULLS:  JP      START
DB      'Z3ENV'         ;identifies program for ZCPR3x
DB      1               ;external environment
Z3ENV:  DW      00000H          ;this address set by Z3INS or
ZCPR3x
DW      NULLS           ;for Type 3,4 compatibility
;-----

;configuration area
DB      'NULLS',0       ;PRGM ID FOR ZCNFG
DS      10              ;extra space, just in case..
DEFNUL: DB      16
;-----

START:
ld      (stack),sp      ;save CCP stack pointer
ld      sp,stack        ;set up local stack

ld      hl,tbuf         ;CCP puts the command tail here
ld      a,(hl)          ;number of characters in the tail
or      a               ;any argument?
jr      z,default       ;no, set default nulls if possible
inc     hl              ;point to first character
call    sksp            ;skip over any space characters
ld      a,(hl)          ;get the first non-space
cp      '/'             ;info request?
jr      z,help          ;yes, if Z
cp      '?'             ;this also gets info
jr      z,help
call    eval            ;get the argument in E (SYSLIB
routine)
call    nc,gsnulls      ;set nulls if arg is valid
;fall through and do nothing on error

exit:
ld      sp,(stack)      ;restore CCP stack pointer
ret                     ;exit to CCP
;-----

default:
ld      a,(defnul)      ;get default nulls from config area
ld      e,a
call    gsnulls         ;and set nulls
jr      exit            ;all done

;-----
gsnulls:
;entry  E = number of nulls to set
;exit   if E was 0ffh
;               A = current nulls value
;       else nulls set to requested value

push    de
call    byetst          ;ZREMOTE/BYE present?
pop     de              ;E contains number of nulls
ret     nz              ;nothing there to set!
ld      c,72            ;extended BDOS function call,
;defined in BYE and ZREMOTE
call    bdos
ret
;-----

byetst:
;routine to test for presence of BYE/ZREMOTE
;entry  none
;exit   flags Z if ZREMOTE or BYE present,
;       else NZ

ld      c,20h           ;ZR & BYE intercept this function
ld      e,0f1h          ;which is set/get user with an
;illegal value in E
call    bdos
cp      'M'             ;..and returns ASCII M if there.
ret                     ;Z=>BYE or ZREMOTE is present
;-----

help:   call    print
db      'Sets Nulls in BYE or ZREMOTE',cr,lf
db      '  SYNTAX:  SETNULL <ARG>',CR,LF
db      '       <ARG>   result:',cr,lf
db      '       none    sets 16 nulls',cr,lf
db      '       number  sets number (0...254) nulls',cr,lf
db      '       / or ?  gets this screen.',cr,lf
db      0
jp      exit

ds      10*2            ;allow 10 level stack
stack:  ds      2
end             ;marks end of code for assembler
```

```
ZML ZREMOTE.COM=ZREMOTE,MYDRVR,ZREMOTE.HDR/I P
```

Here, the /I tells ZML what to do with ZREMOTE.HDR, and the 'P' argument specifies that the rest of the image is to be made as a PRL module. (ZML creates a PRL header if the name of one is not supplied.) That's it! Type ZREMOTE, and it will sign on and become invisible. Your computer works like it always did, except that you can connect a second terminal to your auxiliary port. It will operate in parallel with your main console. Or you can connect another computer (with an RS232 cable) to that port and use a program like MEX or IMP in that machine to communicate in terminal mode or file transfer mode. That's a simple two-station LAN! There is no program with ZREMOTE03 for removal of this RSX. You can use Bridger Mitchell's REMOVE.COM if you have it, or you can obtain KILLZR from Z-node Central (213-670-9465).

Conclusion

That's it for this session. As you can see, the easy part of AL programming is writing code! The *hard* parts are the same as in HLL programming: knowing the computing environment and planning the program to accomplish well defined objectives.

This second part was written before the first was published, so questions you have cannot be addressed. I will, by the time you read this, have talked to many readers at the Trenton Computer Festival. The next article will attempt to address issues that your responses and theirs bring to light.●

References

11. Maley & Earle, The Logic Design of Transistor Digital Computers
    pub. 1963 by Prentice-Hall, Inc
    LCCCN 62-19494
12. Bridger Mitchell, Advanced CP/M - Extending the Operating System
    TCJ, Issue No. 34, Sept/Oct 1988, p. 30
13. Harold F. Bower, LINKPRL - Making RSXes Easy
    TCJ, Issue No. 40, Sep/Oct 1989, p. 16 (part 1)
    TCJ, Issue No. 41, Nov/Dec 1989, p. 27 (part 2)
14. Jay Sage, The ZCPR3 Corner - PRL files and Type-4 Programs
    TCJ, Issue No. 34, Sept/Oct 1988, p. 20
15. B.W. Kernighan and P.J. Plauger, Software Tools
    Pub. by Addison-Wesley, 1976
    ISBN 0-201-03669-X

Sources:
For CP/M manuals: Elliam Associates, P.O. Box 2684, Atascadero, CA 93423

For ZSDOS/ZDDOS, ZCPR34, ZMAC/ZML/ZMLIB, other Z-System programs: Sage Microsystems East, (See advertisement on back cover of this issue)

Real, from page 47

Harvard architecture, with separate paths for instructions and data. This prevents CPU bottlenecks without requiring the programmer to make any special efforts. The '532 and 'GX32 processors have 1.5 kilobytes of on-chip cache memory, but it's divided into two two-way caches. This basically means that there are four caches, two for instructions and two for data. Notice that if you had only one big cache, like the 486, data references could fill all but one of your cache sections, causing cache misses on every subroutine call and return. This is very likely, too, because one cache section will almost always map to the stack. The NS32 design ensures a better hit/miss ratio, you might say, the chips maximize your "cache flow."

Because the NS32 chips use memory-mapped I/O, logic has been implemented to prevent I/O locations from being cached. The 486 does not have this feature, so watch out if you plan to use memory-mapped I/O. Also, DMA logic or multiple-CPU systems might involve changing memory contents outside the CPU, so cache invalidate instructions are needed to prevent the old data from being reused. The NS32 chips allow you to invalidate either a whole cache or only a small part of it. The 486 only allows the entire cache to be invalidated, causing immediate cache misses. You might call this "cold cache".

The newer Motorola CPUs feature on-chip caches as well. Now that you know what the terms mean, you can assess their claims for yourself, because I'm out of space for this topic - but I'm not out of puns yet. I had these left over: "take it in cache", "cache and carry", "cache crop", "cache in on" ...

In summary, Intel's 486 cache is a big, and welcome, improvement, but they've still got a lot to learn. While I'm on the subject of Intel 386 and 486 CPUs, as one whose work obliges him to work daily with these critters, I have to point out how Byzantine these chips are. Backward compatibility to the 8086 and the 80286 is accomplished by a set of modes layered atop segments, with global and local descriptor tables, layered atop a paging scheme. The result is bizarre to the point of being schizophrenic. For example, 32 bit instructions will work in "real" mode, but some 16-bit instructions won't execute in 32-bit mode. In 32-bit mode, however, code segments may be flagged as containing 16-bit instructions, and the chip will change modes dynamically. And then there's the "virtual 8086" mode. It's no surprise that most 386/486 users just pick one mode, usually either the "real" mode or the "32-bit small-model" mode, and pray they don't get bitten by the others. It makes me wonder just how many of the thousands of transistors are dedicated to putting the user through such torture.

The Pi benchmark

A new benchmark circulated on Usenet calculates the value of pi to up to 15,000 digits. It's fun, simple, and uses a preponderance of long math. It's just the thing to make people realize how horribly slow their PCs are and how fast the VAX is. After all, all's fair in love and benchmarks, right?

Here are some timings for 1000 and 10,000 digits for various machines, in minutes and seconds:

CPU:	MHz:	OS:	Compiler:	1000	10000
80286	12	DOS	Datalight	1:19	--
32016	10	Metal	C16	0:47	--
80386	33	OS/2 1.21	Microsoft 5.1	0:39	--
80386	33	DOS	Microsoft 6.0	0:29	--
Microvax-II	?	VMS	GCC	0:27	--
68020	?	Sun 3/60	Sun	0:22	--
32532	25	-	GCC	0:06	9:52
80486	25	RMX-III	iC 4.2	0:05	8:12
VAX 6410	?	VMS	GCC	0:03	5:48

The interesting thing about this benchmark is that it's virtually all long math (no floats are used). For this reason, 16-bit environments pay a heavy penalty. The strong showing by the 80486 is mostly due to running in 32-bit mode, but the cache and optimizing compiler help too. As for the 32532, we're still tweaking it.

Next time

Next time we'll examine the 32CG160 and some other interesting industry trends. In the meantime, speak softly ... and carry lots of cache.●

```
   OC1F TMSK1 C!
   IF RIGHT ELSE LEFT THEN
   EI
;

: !STEPCNT ( StepCnt MCB – ; Set Step Count )
   STEPCNT + !
;
: !+SLOPE ( +slope MCB – ; Set +slope )
   +SLOPE + !
;
: !-SLOPE ( -slope MCB – ; Set -slope )
   -SLOPE + !
;
: !CV ( CV MCB – ; Set Constant Velocity )
   CV + !
;
: !InferenceTable ( InferenceTable MCB – ;)
                  ( Set State Knowledge Base )
   InferenceTable + !
;

: STOP_MOTOR ( MCB – ; Stop the motor )
   >DONE_KB @ SWAP !InferenceTable
;

( Debugging tools to help us develope more )
( wonderous applications!! )
CODE STEP_CYCLE  ( MCB – ; Step thru one inference cycle )
ASSEMBLER
   0 ,Y  LDX  INY INY

   PSHY
   InferenceTable ,X LDD
   ^InferenceTable ,X STD
   Engine ^ JSR
   PULY

   NEXT ^ JMP
END-CODE


: DUMPMCB  ( MCB – ; Aids in debugging. )
CR
." ACC  RATE ACCL STEP +S   -S   IT   ^IT "
CR
DUP ACCUMULATOR + @    0 <# # # # # #> TYPE ."  "
DUP RATECNT + @        0 <# # # # # #> TYPE ."  "
DUP ACCLCNT + @        0 <# # # # # #> TYPE ."  "
DUP STEPCNT + @        0 <# # # # # #> TYPE ."  "
DUP +SLOPE + @         0 <# # # # # #> TYPE ."  "
DUP -SLOPE + @         0 <# # # # # #> TYPE ."  "
DUP InferenceTable + @ 0 <# # # # # #> TYPE ."  "
   ^InferenceTable + @ 0 <# # # # # #> TYPE ."  "
CR ;

: stepIt ( MCB – ; This steps the Inference Engine )
         (        ; and displays the MCB structure  )
   DUP STEP_CYCLE DUMPMCB
;
```

Listing 2

```
CREATE OTHER_MCB MCB_SIZE ALLOT
                 ( The MCB we will use for the NMIS)
                 ( NMIS 7040 Stepper motor example.)

OTHER_MCB          ( MCB )
doState            ( Action Routine              )
>SENSOR_ACTION @   ( Sensor Routine              )
0                  ( Reload Timer count "reserved")
8000               ( Address of Motor Port       )
04                 ( Address in Port of Motor    )
INIT_KB            ( KB for starting up inference )
100                ( constant velocity count     )
100                ( step count constant         )
5                  ( acceleration constant       )
-1                 ( deceleration constant       )
ADD_MOTOR          ( add this motor to list!     )
```

Stepper, from page 13

Bases, the Inference Engine is compiled along with its support routines.

The Engine

The Engine is actually quite simple. It is given a pointer to a set of rules. It cycles through each rule, executing the condition portion of the rule. If the condition portion of the rule returns with the CC register Z bit set (We'll call this FALSE), the Engine will call itself through a recursive call. When the condition portion of the rule returns with the CC register Z bit clear (We'll call this TRUE), the Engine fires the TRUE condition's action and completes its cycle. The action portion of the rule alters state variables in the MCB and can initiate external events such as "Step the motor". The above cycle is termed the "Inference Cycle".

Rule priority is governed by the rule's placement in the State Knowledge Base. One rule MUST fire its action in the active State Knowledge Base. This is not a caveat because of the execution speed gained and the nature of state machines. It does mean that all of a state's possible events must be carefully and methodically thought out.

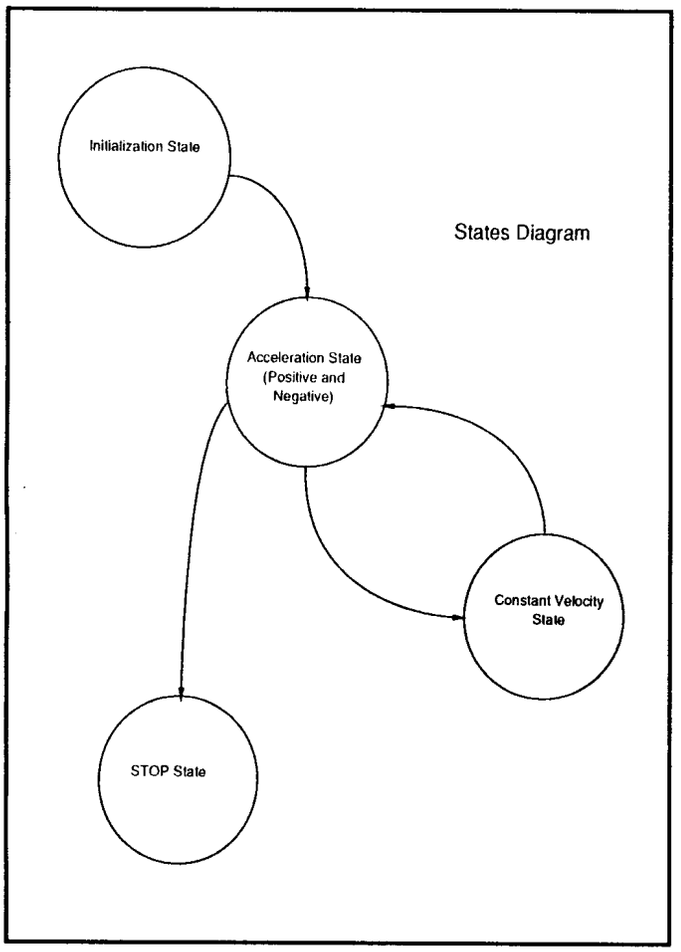
Care must be exercised when designing the rules which fire during an inference cycle. Inference Latency is the time required to complete one inference cycle as it relates to the current timer interrupt interval. If the inference latency is longer than the interrupt interval for one cycle, the regularity of timed interrupt is broken and the system's confidence degrades dramatically. In some instances a "crash" can occur. If more than one device is attached to the timed interrupt, the complexity involved in the determination of maximum inference latency increases. Once again it is evident that the states must be factored into rules concisely and methodically, and the implementation of rules must be carefully thought through.

On Real-time Systems

In a real-time system model, various events are monitored. When specific events occur actions are taken. These actions either directly or indirectly alter the system's behavior. Events and their reactive actions can be grouped together. These groups are states. A state has a specific goal. When the state goal is achieved, a new state or behaviour is established. Along with the new state a new goal is asserted which governs the state. Some action in the retired state establishes the next state to be activated. The state transition diagram and the transformation specification are two structured design tools that enable us to quickly and concisely determine the required rules for the various states. Figure 1. defines the required states and their relationships. Figure 2. is the state transition diagram of our stepper model. (At this point the reader may wish to refer to article II. of this series.) Figure 3. is the derived transformation specification. The transformation specification maps directly to a system of rules! The implementation details can be directly derived from this system of rules.

The Interrupt

Within the timer heartbeat interrupt that propels our system, the Y register is used in two layers. The interrupt cycle uses the Y register to hold the current MCB, and the Engine uses the Y register to cycle through the current State Knowledge Based. Care is taken to ensure the integrity of the Y reg-



Promotional
and
Technical Writing
for Electronics Marketing
★ ★ ★ ★ ★
Technical Articles for Publication
Advertising Concepts and Copy
Product and Service Brochures
Press Releases
Speeches and Lectures
Editing/Rewrite Service
Consulting
★ ★ ★ ★ ★
Bruce Morgen
P.O. Box 2781
Warminster, PA 18974
215-443-9031
(Voice, Data by Appointment)

ister for its subsequent use by the non- interrupt level Forth system. (The Y register is used as the Forth data stack pointer.) doState maintains the integrity of the Y register as used in the interrupt cycle. Once Engine is finished, the Y register is restored to its function of MCB pointer. Once the interrupt is complete, the Y register is restored to its function of being the Forth data stack pointer.

Tools

Certain tools are required to allow experimentation, and debugging if necessary. ADD_MOTOR adds more information to the MCB now. The code illustrates its usage. RUN_MOTOR is now MCB specific. It also accepts a flag to determine if rotation is to the Right or Left (TRUE is Right).

STEP_CYCLE steps the inference engine one inference cycle. It grabs the current State Knowledge Base and sets the appropriate registers for the Engine. It then runs the Engine for one Inference Cycle. DUMPMCB displays the important members of the MCB object. stepIt combines the two to form a single step mechanism for debugging rules. Set up the appropriate (and or troublesome) set of events and "stepIt" to see why the rule set in question does not work.

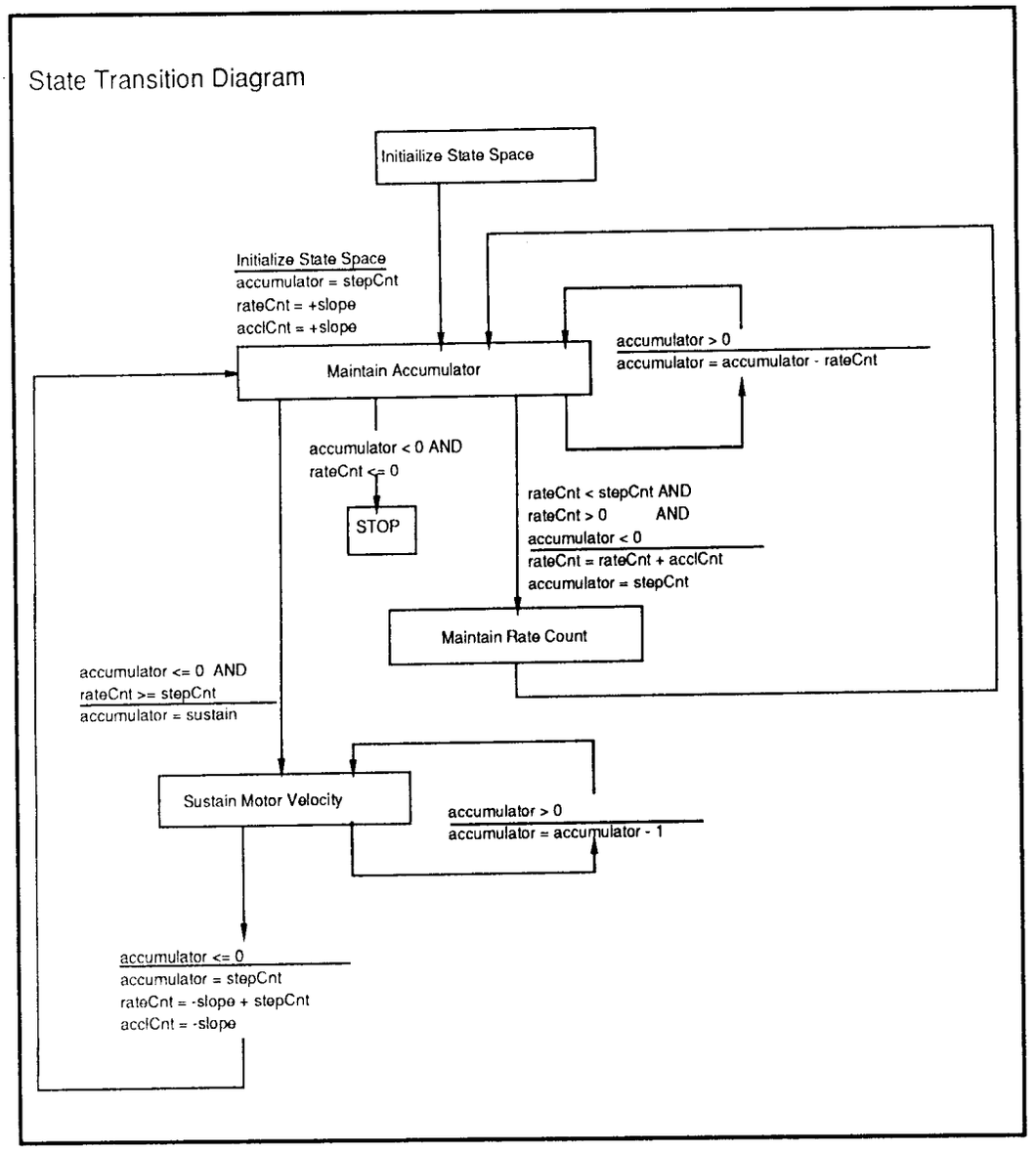
Figure 3

```
1.0             Initiailize State Space

        accumulator = stepCnt
        rateCnt = +slope
        acclCnt = +slope

2.0             Maintain Accumulator

IF
        accumulator > 0
THEN
        accumulator = accumulator - rateCnt

2.1             Maintain Rate Count

IF
        rateCnt < stepCnt       AND
        rateCnt > 0     AND
        accumulator <= 0
THEN
        rateCnt = rateCnt + acclCnt

3.0             Sustain Motor Velocity

IF
        rateCnt >= stepCnt      AND
        accumulator <= 0
THEN
        rateCnt = rateCnt + acclCnt

3.1             Sustain Motor Velocity

IF
        accumulator > 0
THEN
        accumulator = accumulator - 1

3.2             Sustain Motor Velocity

IF
        accumulator <= 0
THEN
        accumulator = stepCnt
        rateCnt = -slope + stepCnt
        acclCnt = -slope

4.0     STOP
IF
        rateCnt <= 0    AND
```

Try variations on TIMER_OFFSET, STEPCNT, +SLOPE, -SLOPE and CV with the !CV, !STEPCNT, !+SLOPE, !TIMER_OFFSET and !-SLOPE words. Remember that these words are MCB specific; an MCB MUST be on the stack before calling these words as well as the instantaneous value. Experiment with your own rules by using the !InferenceTable word. !InferenceTable establishes a State Knowledge Base for the identified MCB. !InferenceTable is also used to restart a motor. Call this routine with the appropriate Initialization State Knowledge Base and the selected motor will restart.

Two'fer

Listing 2 illustrates two motors in place by the addition of a second motor. Note how simple it is to add a second motor. All that is required is an MCB sized block of memory! The rules are "reentrant" in that they have no knowledge of individual motors, only knowledge of how to run a motor. All instance information or contextual information resides in the motor's MCB.



On Stepped Inference

Stepped Inference is a way of implementing complex state machines and state intense models. It can be used for other applications from communication protocols to neural networks. I have found that this approach saves space, time and confusion on these sorts of problems. You may find this to be so in your endeavors.

Next time I hope to have a feedback mechanism to play with. We will talk about the lack of torque that occurs in steppers at higher RPMs and how to counter that with Boost techniques. Until then, have fun with Forth! ●

LAN, from page 32

knowing a whole lot about it. You don't know how long the message will be or whether it all got there in one piece. There may be errors to deal with.

So it should be easy to see that a transmit process (program) is normally much smaller than a receive process, with the result that almost every machine can talk faster than it can listen.

When all machines are about the same in capability, though, that is not too much of a problem because the average sending rate of any one machine will not exceed anyone else's average receive rate (often machines alternate sending and receiving, so your own receives will moderate your transmit rate).

If one machine is supposed to be used by many other machines, there is a design problem. If it is made to be able to talk fast and do so continuously so that it can service many users, its average transmit rate will greatly exceed the average receive rate of the users.

This causes a problem called retransmissions, where a receiving station was processing a previous message and missed this one, causing the sender to time out and send again. On the other hand, if the average transmission rate of the central unit is made to match the average receive rate of all users, then no one will be serviced often enough, giving users the impression that the network is slow.

In summary, three major reasons for poor LAN performance are: rules violations; excessive broadcasting, which steals cycles from all users; and retransmissions, which in effect cut the signaling rate by a factor of the number of retransmissions needed.

Next time we will look at some of the implementation details of Ethernet and its CSMA/CD (Carrier Sense Multiple Access/Collision Detection) method of communication.●

Computer Corner, from page 64
the market is going. Products like the soon-to-be-released Deskview X-window will certainly be one of the front contenders for a new standard. If I were developing a new product, OS2 however would not be a platform I would write it for. It would be very modular, with the user interface replaceable in case my current choice doesn't make it in the market.

Next Front

The area where I do see a major change and plenty of growth, especially for the small developer, is embedded systems. To develop anything for the new and bigger boxes, requires a large financial investment. I said before how to test the program I am currently working on required 30 PC workstations in a LAN based network. Each station cost about $5000 of OEM money (large discounts). Not many garage developers can spend that kind of money to get their product into the market.

Embedded systems and small PC-based development can still be done by the little person. The cost of small controller boards can be very inexpensive to design and build. The average small garage could easily hold the needed material and space to build hundreds of 6805 or 8048 controllers. Special software that could turn a small PC-XT into an industrial controller is where I think the major chance for growth over the next few years will be.

The competition for the major systems is now in the hands of multi-million dollar companies who can afford the problems and time needed to come to market. This has also forced out the little mechanical engineer who needs a small controller. Will they be willing to play with the big boys and their high up front cost. I doubt it. The little guys want little developers. They want a personal approach, ability to talk with the person doing the work. It will be this hand holding of the clients that makes developing small systems work. I feel we have not began to see just how much this market is capable of supporting, *yet*!

Cross-Assemblers

In embedded development, cross-assemblers are the name of the game. I have recently become involved in the search for a new 68K cross-assembler at work. The story goes like this: used Motorola exormacs development system for ten years; need to move to PC based systems; find compatible cross-assembler for 68K and 68K "C" existing code; try a number of assemblers; reject most but find one; fix up code to work with new choice; now do all work on PCs'. Well that project is still going and will go for another year. We have tested the cross-assemblers and settled on Microtec. I like Avocet assembler better, but their macro pre-processor caused too many problems. Avocet 68K "C" was so bad I can not recommend you even consider it for testing.

We selected Microtec for two reasons, no major code bloom in "C" output and a one-pass assembler that did most of our macros without change. The previous programmers used all undocumented and not-allowed operations inside the macros. This makes the macros almost impossible to port to new assemblers. The "C" code was done on a Greenhills Unix product that is not available for DOS use. It produced very tight output, which fit tightly inside our ROMS. We tried the Avocet "C" compiler and it produced almost 30% more output for the same source files. The Microtec is only 5% larger, still not good but with a little trimming of options we can fit it in the ROM.

What was interesting is how others have reviewed the cross-assemblers. An article came during this time, and all the reviewer was interested in and tested for was speed. Speed was the least of our concerns. Compatibility and code output size are top on our list, with speed on the bottom. Actually my personal most important item is how well it produces *correct* code. I don't care how fast it is; if it produces code I can't trust, who needs it.

Porting

A few projects floating around my place are porting operating systems. I am finally getting back to my Sage system. This 68K system needs a new operating program and I plan to port Minix to it. Until I get Minix ready I am checking out porting Forth. The Minix will take some time with all the files needing changing. Forth, on the other hand, should be quicker by using the public domain programs from GEnie.

I have down loaded a new program that may be worth looking at. M16PC is a complete metacompiler to move F83 to 68K based systems. The code by Wilson Federici is currently set to work on a PC and produce Atari ST-compatible F83. The main thing is how you can make minor changes and be able to output for your own projects. Metacompiling has been around from day one of Forth; the problem is getting enough information on how to use it. Here we have a working program so one only needs to change the I/O screens, then compile and test. I have worked with Mr. Federici before and he does good work, so his port will also provide some good samples of how to do it.

I plan on doing this for the Sage as soon as I can and will have more later. My main plan is finding a way to put F83 into ROM on 68K systems (also Z80). I have lots of old machines with limited memory and Minix will not work in those places. So that leaves Forth as the only way to make use of all these different and limited machines I have.

'Till Next Time

Well that is about it for now, and hopefully I have made you consider some alternatives for the future of computing. I still keep coming back to doing things in Forth, especially after fighting with cross-assemblers. Hope your fights are little ones!●

TCJ On-Line

Readers and authors are invited to join in discussions with their peers in any of three on-line forums.

- GEnie Forth Interest Group (page 710)
- GEnie CP/M Interest Group (page 685)
- Socrates Z-Node 32

For access to GEnie, set your modem to half duplex, and call 1-800-638-8369. Upon connection, enter HHH. At the U#= prompt, enter XTX99486,GENIE and press RETURN. Have a credit card or your checking account number handy.

Or call Socrates Z-Node, at (908) 754-9067. PC Pursuit users, use the NJNBR outdial. StarLink users, use the 3319 outdial.

Z-Best, from page 45

quite frequently, but whose power is often taken for granted. Based on FINDF vs 2.6 by Bruce Morgen, FF finds all files matching a list of file specs on all drives, a specific drive, or a list of drives. See *Figure 4* for FF's syntax.

Figure 4

```
Name          Vers   S ZSUS Siz Rec CRC  Library/Size Issued   Author
============ ====== = ==== === === ==== ============ ======== ================
FF.COM        2.40   4 V205   4  31 A609 FF24        73 01/19/91 Al Hawley
File Find utility which finds all files matching a list of file specs on all
drives, a specific drive, or a list of drives.  Vs. 1.0 (3/87) by Jay Sage.

Syntax:  FF [D: or DIR:]afn[,afn]... [d...][/o...]                          ]
```

All file specs are automatically made wild, so that "A.B" becomes "A*.B*". The "d" before the slash is the list of drives to scan. The "o" after the slash can be E – exact matches only, P – no paging of output, and/or S – inclusion of SYS files (based on configuration defaults).

Al Hawley has made some significant improvements to FF which make it both noticeably faster in file searching and easier to install:

1. FF has been recoded a) to use SYSLIB43C routines, including the new and faster sort routines developed by Joe Wright, and b) for brevity, clarity, and code optimization.

2. The filename parser has been changed to accommodate both DU and DIR forms in the list of search filenames. Adding a properly working NDR (named directories) function to FF was no small task. Al had to write a routine to search the NDR after failing to find such a function in Z3LIB (are you reading this, Hal?). Neither was there a function available for returning or testing the password. With FF24, two drive search vectors can be configured—one based on the currently logged drives and the other based on drives represented by entries in the current NDR. By subjecting use of NDRs to the status of the wheel-byte, you can restrict search to NDRs for non-wheels and within certain drive-map constraints (a much needed security feature if you're allowing use of FF on a RAS!).

3. The command-line syntax has been relaxed to allow white space after the drive list, and the "/" before the option field may also be replaced with white space.

4. Many users of earlier versions of FF have had difficulty patching the drive table to skip certain drives on their system. Gene Pizzetta at one point (vs 2.1) provided a patch file (FFPATCH.ASM) to edit/assemble/overlay to FF which simplified the process, but version 2.4 now makes this quite effortless. A ZCNFG configuration file (.CFG) displays a table of possible drives (A-O) in which one simply toggles the desired search drives. A candidate set of drives is even presented as a default. This is made possible by FF's use of your Z3ENV drive vector (if any) and a mask derived from your Z3ENV maximum drive to record your current system's drives. The Z3ENV maximum user, by-the-way, is employed only for non-wheels. A wheel will get a search of all 32 possible user areas.

5. Defaults for the three command-line non-drive options (E, P, S) can also be configured with ZCNFG.

The development issue now being tackled with FF is the protocol for reporting data returned by FF. Should FF's function be expanded from simply finding and reporting a list of files to one of doing something with that list (writing the list to a file, deleting the files found, etc.)? Certainly Z-System offers ways of manipulating the output of such programs. FF has been used innovatively on some RAS's, for example, to ask a user if he would like a description of the file it has found. The count of files found is placed in a register and checked by an alias which then poses the question at the right time. By setting the error flag, FF even saves the correct count if the user aborts a search (the desired file is found and he doesn't want to waste time searching the remaining drives). Writing a file count to a register is easy (EXFIND.COM, another RAS tool, uses the same approach to report the existence of files in archive catalogs); reporting a filespec (DU:filename) to the environment, however, is more complicated. The question is whether FF and similar tools (AFIND, XFOR, to name two) should really be in the business of redirecting output to a file and other programs. We'll keep you posted on any decisions in this area which may drive further development of FF.

Waving the Z Flag...

John Dvorak in PC Magazine last April described the MS-DOS version of the now ancient CP/M text editor VDE thus: "This may be the finest piece of word processing code ever written. I've never been as impressed with anything as I have with VDE." Of course, VDE has been even more improved upon by Carson Wilson, whose ZDE might now be the most used program under Z-System. What astounded Dvorak most was that VDE is a full-blown word-processor in just over 40k of code! To me, this is a great testament to the skill and diligence of CP/M programmers in continually improving their work, not only in terms of functionality, but of efficiency of design and, most importantly, of usability for as wide a range of users as possible. VDE/ZDE became works of art due to the care and pride of the author in his work and his continuing interaction with users for feedback and suggestions for improvement. You rarely see this attitude in the MS-DOS world (at least not to the extent that it exists in CP/M), and the result is that tons of sloppy, bloated code are dumped daily into the MS-DOS Public Domain with little or no support and usually with the main goal of making money, not sharing ideas with and supporting the wider community of users. Even beyond the sharing of ideas and source code, though, there is a set of values at play in the CP/M world which I think is right and which has kept me involved to this day. Maybe at the heart of it is just a simple feeling of belonging, of sharing in a commitment to an ideal ("big is not necessarily better") and in the challenges that has brought us. Whatever it is, you can't beat the camaraderie that develops so easily among CP/M'ers. What's not to like about someone who understands where you're coming from, respects your opinions, values your suggestions, and gives freely of his time and energy when you're having a problem? You've undoubtedly heard this "spirit of CP/M" spiel before, but, believe me, based upon my experiences with the group of Z-System developers I've known over the past five years, this spirit still exists. Get yourself a modem, connect to one of the various Z-Nodes and follow the message bases. A list of the nodes is on the outside back of the shipping cover that protects your *TCJ* in the mail. You'll find that the spirit is very much alive and evident in the help and support available for anyone who seeks it.●

IOP, from page 40

TurboROM code, hidden from view. TurboROM typeahead buffering takes place on this level and must itself involve a fairly sophisticated relationship between CONST and CONIN to maintain the typeahead feature. In the view from the RAM BIOS, however, this relationship is invisible.

At the level of the RAM BIOS, CONIN and CONST must cooperate to manage the single-character lookahead required by the function keys. At the level of the IOP, these functions cooperate to extend the function of the number keypad keys, the cursor control keys, and the grey keys above the number keypad, which I frequently refer to as extended input keys.

The BIOS code in Listing 1 should be fairly self explanatory. Note that the one-character lookahead in the BIOS is supported by self-modifying code. A true/false flag in CONST (bufclr) works with a one-character buffer in CONIN (bufchr) to determine whether the ROM BIOS routines for character input are called or a previously stored character is used as the next available character input. The TurboROM's single-key mapping for number keypad and cursor control keys is done by the mapfnc routine. Note that the output from this routine is switchable depending on a TurboROM configuration bit. If no IOP is installed, mapfnc maps these keys to the values listed at akeys and keypad. When the IOP is installed, this configuration bit is reset (by the IOP's tinit routine), and the key codes returned by the TurboROM are passed unaltered to the IOP, where they are used to select extended input strings, as explained below.

The IOP

The relevant portions of my custom IOP are shown in Listing 2. With no IOP or the NZCOM dummy IOP installed, most of my normal, factory installed keys return their marked values, and my added keys are effectively inoperative. When my IOP is installed with NZ-COM or JetLDR, the tinit routine is called to perform a number of basic installation and initialization functions. (Because most of these functions are common to all IOP's, the tinit routine is not shown in Listing 2.) Briefly, tinit substitutes jumps to its own routines for the 7 BIOS jump-table entries dealing with console I/O and the list, punch, and reader physical devices. In my IOP, all but the CONIN and CONST routines simply redirect program control into the BIOS proper. Additionally, my tinit must turn off the mapping of number keypad and cursor control keys (extended input keys) by the BIOS so that the IOP can assume and expand on this task.

Although I've added four "grey keys" above my number keypad to the set, the extended input keys on the original Kaypro 10 consisted of only the cursor control and number keypad keys. The mapping data for these keys was read from the system tracks and was, while the system was up and running, memory resident. These keys could be remapped easily, either temporarily (through modifying memory) or permanently (by writing the necessary information to the system tracks). Software such as the original Kaypro CONFIG program and the terminal initialization routines in the Kaypro-installed WordStar took advantage of this facility.

```
        jr      nz,frjp     ; If so, frjp will return one.
        ld      a,(iobyte) ; Otherwise, as usual ......
        rrca
        jp      nc,reader

; Retchr will contain the return value from one of the grey
; extra number keypad keys, if one has been pressed.
;
retchr  equ     $+1         ; Where exkeys go
        ld      a,0
        ld      c,a         ; Retchr to C (juggling act)
        xor     a
        ld      (retchr),a ; Clear retchr
        ld      a,c         ; Retchr to A
        or      a
        jr      nz,isexkey ; If retchr > 0, Use exkey
                            ;  value instead of conin
                            ;  return.

        call    conin       ; If no retchr, go get char
                            ;  from BIOS.
        or      a
        jp      p,outkey    ; Msb not set, so we have an
                            ;  ASCII key.

; Isexkey is executed if a key from either the number keypad
; keys, the cursor control keys, or the extra grey keys above
; the number keypad has been pressed.  Both isexkey and frjp
; are supported by a table of strings in high memory of from
; 1 to 4 characters each, the address of which is returned by
; the special BIOS routine calspc.

isexkey:
        and     01FH        ; Form index to table
        ld      c,4         ; Get address of keytable
                            ;  from BIOS
        push    af          ; Save off character in question
        call    calspc      ; Calspc returns with extended
                            ;  keytable address in HL
        pop     af
        sla     a           ; Shift left twice to
        sla     a           ;  multiply by 4
        ld      c,a         ; Get new character position
        ld      b,0
        add     hl,bc       ; Set hl to character position
        ld      (pntr),hl  ; Store position in pointer in
                            ;  code below
        ld      a,4         ; Set to this value
        ld      (cnt),a     ; Initialize in-code counter to 4

; Frjp is executed after isexkey and while an extended input
; key character remains in the pipeline.
;
pntr    equ     $+1
frjp:
        ld      hl,0000     ; Get pointer value
        ld      c,(hl)      ; Load character into C reg
        inc     hl          ; Increment pointer
        ld      (pntr),hl  ; Save this value off
        ld      a,(cnt)     ; Retrieve current counter value
        dec     a           ; Decrement this value
        ld      (cnt),a     ; Save this value off
        ld      a,c         ; Move character into A
        ret     z           ; Return if char is last of 4
        ld      a,(hl)      ; Get next byte value
        or      a           ; Compare to 0
        ld      a,c         ; Char to A again
        ret     nz          ; Return if intermed. char &
                            ;  next not nul
        xor     a
        ld      (cnt),a     ; Clear counter value
        ld      a,c         ; Char to A once more
        ret                 ; Return if intermed char &
                            ;  next is nul


        .
        . (remaining IOP code)
        .
        end
;
; End of NZIOP.Z80
```

The original Kaypro BIOS provided the ability to assign strings of up to 4 characters to each of the extended input keys, but this capability was not continued in the Kaypro TurboROM and its RAM-resident BIOS. Several years before my purchase of a TurboROM, I had developed the KEYSET software which allowed these keys to be mapped to predefined character strings from a single command line, and by the time I converted to a TurboROM, I had a considerable programming investment in application macros and keypad configurations which took advantage of this flexibility. I was unwilling, therefore, to settle for the TurboROM's single-character key map, even though remapping was still a fairly simple software task. Once I understood the IOP well enough to implement my function keys, it was only logical to extend my code somewhat to enable the IOP to support the old Kaypro character strings of which I had grown so fond.

Once my IOP has been installed using NZCOM, it intercepts all calls to CONST and CONIN. Calls to CONST, either directly or from within CONIN, check the value returned in the L register by the BIOS CONST, and, if it is nonzero, an appropriate system function is executed. Otherwise, CONST simply returns 0 or 0ffh in a normal fashion. IOP CONIN enhancements are concerned primarily with the extended input keys, and, like the original Kaypro BIOS from which the code is adapted, enable up to 4 characters to be returned from a single keystroke.

Note that the grey keys above the number keypad constitute a special problem. These keys are not logically function keys, since they return characters instead of executing system functions. However, they are passed to the IOP by CONST as function keys, i.e., with A = 0 and with L equal to the raw SIO input from the keyboard. This problem is circumvented with another inline code modification. If a grey key has been pressed, the IOP CONST assumes that it is a function key, the function of which is to poke it's value into retchr in the IOP CONIN routine so that the next call to CONIN will retrieve it instead of calling for input from the BIOS CONIN.

The jobs I selected to have performed by my extra keys were chosen because of the special needs of my system—my MicroSphere RAM disk and print buffer, my TurboROM, and my attachment to the Kaypro extended input keys. One might easily choose a radically different set of functions, which could be as easily coded into an IOP as those I have chosen. A screen recorder might be toggled on and off or a byte entry mechansism might be installed which allows one to type in the ASCII value of a desired character following a special key, as on all MS-DOS machines. Because IOP modules are interchangeable, it would be easy to swap one set of functions for another using NZCOM or JetLDR. There are also 6 more key positions on the stock Kaypro 10 keyboard, 3 each on either side of the space bar, which are available for use.

Few limitations exist on what one could do with such CP/M function keys. One caution, however, must be observed. Many, if not most, CP/M DOSs, including DRI's original BDOS, are not reentrant. Since CONIN and CONST are usually called via the BDOS, these functions may not themselves invoke any BDOS routines. Any secondary I/O performed by IOP routines must be handled directly by the BIOS. Some of the new BDOS replacements, such as ZSDOS, claim to be reentrant, so this limitation may not apply to these systems. However, other BDOS replacements which claim to be reentrant, such as some versions of ZRDOS, don't hold up well under the strain, or so I've been told.

I will be happy to provide anyone interested in pursuing this subject further with the complete code for my BIOS and IOP. The BIOS code is intimately intertwined with the support code for my RAM disk, while the IOP code contains a number of experimental (and rejected) serial printer routines, extensively commented out. Nonetheless they should prove helpful to anyone seriously interested in fully implementing extra keys in the same manner that I have. My BBS Z-Node 77 phone number is 512-259-1261, and I'll be glad to try to answer serious questions relating to the material I've set forth in this article.●

Editor, from page 2

Yet Another What?

Ever hear of a YASBEC? *Yet Another Single Board Eight Bit Computer.* Paul Chidley, from north of the border, wanted the hottest CP/M box he could make and headed into the workshop. The YASBEC is the result. It runs at 16 MHz, has up to 1 MB of static ram, SCSI interface, the works. Ian Cottrell brought one down to Trenton to gauge the reaction. Some reaction! Paul never intended to go into mass production with this thing. When I talked with him last month, he said all he hoped for was someone to help defray the cost of producing the motherboard. I think Paul needs to make more boards—there is a waiting list already. With luck, we will have something more to tell you in a future issue.

What's With GEnie?

Astute readers will notice that the GEnie advertisement from last issue has been pulled. I have nothing official but from what the rumor mill tells, there has been a tussle between GEnie and a competitor over the use of the term "Star Services." It seems one side feels this infringed on a copyright of theirs. I am no lawyer and don't want to speculate on the merits, but have noticed that GEnie now calls their special service "GEnie Basic."

Be that as it may, the service is the same as before and *TCJ* is on-line in both the Forth (room 710) and the CP/M (room 685) SIGs. You are certainly invited to join us.

Mutual Backscratch

Notice anything different about this issue? No, it isn't printed on edible paper. Count the pages. Issue 47 had 36 pages; today we have 64!

If you recall my first editorial, I said that a journal is written and produced by a small group for people who are called friends. The authors are our treasure. I put the word out that we were seeking good, meaty articles to print and they have responded gladly. I faced a dilemma: should I wait until the circulation warranted an increase in size? Though we are growing fast, that would have meant holding off some of the best articles I have seen in any journal lately. Or should I bite the bullet and move out to 64 pages now?

You see, while *TCJ* is a labor of love, it carries a strong current of reality with it. The larger the journal, the bigger the expenses. Paper costs money, and the more paper you

See Editor, page 58

Z-Corner, from page 36

```
ced syn wow "makewow &a;xwow;ced clear syn xwow"
```

When this alias is invoked, the alias XWOW has not even been defined! The program MAKEWOW creates it based on the parameters passed in the command tail. Then the XWOW alias is run, and finally its definition is removed. I use this complex approach when the program MAKEWOW, which figures out what needs to be done, uses too much memory for it to perform all the tasks itself. The alias that it creates runs later, after MAKEWOW is no longer in memory.

PCALL (Parameter Recall)

With PCED one can declare a list of commands for which automatic parameter recall should be performed. For these commands, the last command line involving the command is stored in a special buffer. The next time the command is invoked without any command tail, the previous command tail is provided automatically.

I have EDIT assigned to PCALL status using the command

```
ced pcall edit
```

When I first started writing this column, I would have used the command

```
edit tcj50.ws
```

Later, I would type "edit" alone on the command line, and the original command would be executed for me. This is a very handy feature but probably not important enough to implement in Z-System, since it would require additional disk activity to save and recall the information. With LSH I would type "ed" and then control-O to recall the last editing command.

[I thought new code would be required to do this under Z-System, but Howard Goldstein showed me that it can be done already! He suggested the following ARUNZ script, assuming that WS is one's editor:

```
EDIT  if -null $1;shvar edtail $*;fi;
        resolve ws $edtail
```

This uses the shell variable facility of Z-System. Storing and then recalling the variable "edtail" would take some time unless the programs and data files are on a RAM disk.]

User Synonyms

So far we have described how PCED functions at the DOS prompt. PCED is also capable of functioning when application programs are prompting for user input. This only works when the input is requested using a particular DOS function call, the one equivalent to CP/M's buffered line-input function (#10). This feature of PCED is normally not engaged (since it could cause problems) and must be activated by entering the command

```
ced on user
```

Once it is activated, many PCED functions become active. The full command line editor can be used, except that command completion using the TAB key is disabled. There is a command history stack that is separate from the one that saves DOS commands. PCED commands can be executed, including those that define, edit, or remove synonym definitions.

Most importantly, one can now define what PCED calls "user synonyms". For example, you might create a display alias for use in DEBUG with the following command:

```
ced usyn d 'D DS:100 L100'
```

User synonyms–defined with the 'usyn' option–and command synonyms–defined with the 'syn' option–are completely independent.

Few of the programs I use regularly get their input via MS-DOS's buffered console input function, and, therefore, I never use this PCED facility. Under the right circumstances, however, I'm sure it could be very handy, and perhaps I should give it a try some day.

Internal (Directory) Synonyms

There is a third class of synonyms that I have found very helpful (and wish I had on my Apollo minicomputer running Apollo Domain or Unix). With its nice, flat named directory structure, the Z-System does not have the problem, but on systems with tree-structured directories, one wastes enormous amounts of time trying to type excruciatingly long directory specifications. For example:

```
cd d:\editors\wordperf\letters\personal\john
```

Tree-structured directories make for a very logical and orderly collection of files, but they sure aggravate finger-tip calluses!

With PCED we could define an "internal" synonym as follows:

```
ced isyn john"d:\editors\
  wordperf\letters\personal\john"
```

Then we could change default directories by entering the simple command

```
cd @john
```

or edit a file there using the command

```
edit @john\file.doc
```

The '@' sign is a signal that the string following it is to be interpreted as an internal synonym name and expanded. I use this mainly for expanding directory names, but it can be used to provide shorthand string definitions for any purpose one wishes.

You do have to watch out for some confusing side effects. Suppose you wanted to edit a file with the name "@johnson.doc" and enter the command

```
edit @johnson.doc
```

The "@john" part would be recognized as an internal synonym, and the command would turn into

```
edit d:\editors\wordperf\
  letters\personal\johnson.doc
```

One has to watch out for all of PCED's special characters. They can generally be made to be taken literally by prefixing them with the variable prefix character (normally '&'). Thus one would enter

```
edit &@johnson.doc
```

There is, unfortunately, a bug in PCED that prevents this

from working in alias definitions. I have pointed this out to the author and hope he will have a maintenance fix. It causes me great difficulty with some aliases for sending Internet electronic mail (where all addresses contain the '@' character).

User Programs

PCED has a very powerful facility for adding independent resident programs. It never occurred to me before, but this is somewhat like the RCP (Resident Command Package) in the Z-System. PCED comes with a number of very nice programs of this type, and software developers can write new ones.

These programs have several advantages. If PCED's user mode has been turned on as we described earlier, then they can be activated while inside an application program. Some of the programs—such as VSTACK, the full-screen history shell—are designed to augment existing PCED features. Other programs are just particularly nicely conceived and written. I will describe only a few of them here.

CDIR is a directory display program whose format I particularly like. It has many options that can be declared on the command line (or automatically in aliases). It can list files in anywhere from one to four columns; it can sort by filename, filetype, or date and in ascending or descending order; it can include only directories, only files, or both; it can display different types of files in different colors for easy identification. It can also generate a file containing the list of file names for processing by other programs.

HS (for "HindSight") buffers screen output and allows one to retrieve it. Now when stuff scrolls off the top of the screen before you could read it, you can get it back! This is particularly usefull on very fast computers, where information can disappear before the finger can respond with a control-S. One can also write the captured screen data to a disk file. In PCED2, HS can be configured to use expanded memory for its buffer, so one can have a large buffer without losing valuable program memory space.

KEYDEF allows macros to be assigned to function keys. Separate definitions apply at the DOS prompt and at user input prompts inside application programs.

KEYIN establishes a buffer which holds simulated keyboard input. When the next prompt for user input occurs, KEYIN will supply characters from its buffer. This allows programs that operate only in interactive mode to be run in batch mode. KEYIN is particularly useful in alias scripts and BAT files.

SEND is a program to send character strings to any device or file. It is similar to the DOS ECHO command but is more flexible. For example, while ECHO always sends a carriage return and linefeed at the end of a line, SEND sends only what you tell it to. This can be very important, such as when sending setup commands to a printer or when constructing a prompt line in pieces.

PCED and Personal REXX

PCED has special interfaces to allow it to work with certain other software packages. I particularly appreciate its coupling to Mansfield Software's implementation of the mainframe batch processing language called REXX.

REXX would be worthy of an entire *TCJ* column, so I will not say much about it here. Basically, it is a programming language for processing strings and generating command lines. Strings can be pulled apart into their words and characters; strings can be concatenated with other strings; strings can be substituted for other strings. All kinds of program looping can be performed. There is a complete interface to the operating system so that one can get the system time and date, check for the existence of files; read and write file contents; change file attributes; and so on. The resulting strings that REXX builds can then be passed as commands to the operating system's command processor.

REXX programs are written in files with the extension REX. Normally one would have to invoke them using the explicit command

```
rexx filename
```

where "filename" is the full path specification of the REX program file. With the PCED add-on program RXRUN loaded, these REX files can be executed automatically, just as BAT files are. In this way, REXX becomes an integral part of the system.

When I described the WOW alias earlier, what I really had in mind was REXX. It takes up over 200K of memory, so one often cannot afford to have it resident when commands are run. To get around this problem, instead of having REXX pass the ultimate command lines to the operating system, I have it pass CED commands that define a new alias. That alias is then run after REXX is finished and no longer resident in memory. Without PCED, REXX would lose much of its power for me.

Well, that completes what I will say this time about PCED. I hope that many of you who use DOS computers will order PCED so that the author will feel it is worth his while to add more of the features that we Z-System users would like to see.●

Editor, from page 56

mail, the more the post office wants. We have the resources to do this, but admittedly, it limits our reserves.

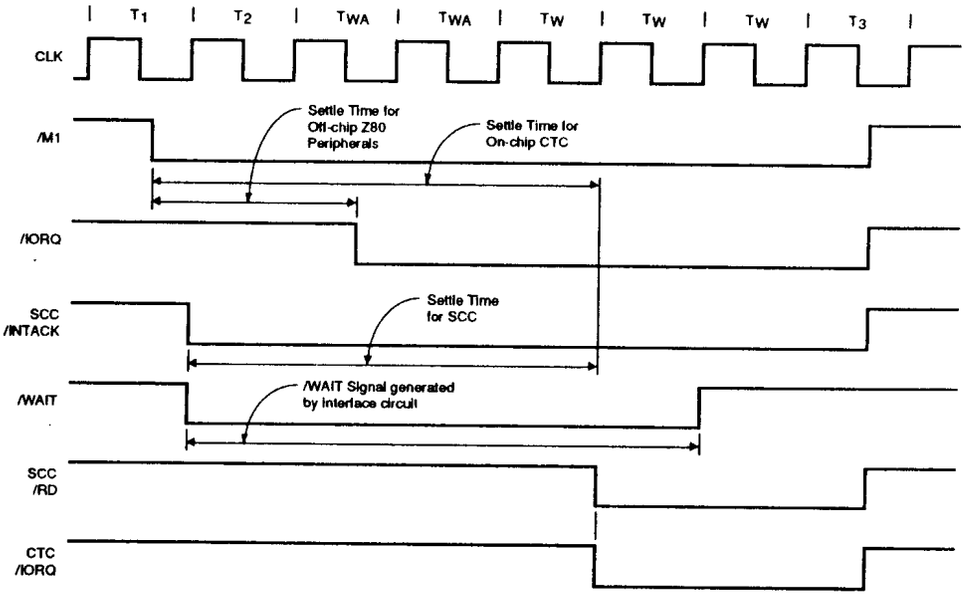
Our authors have made a commitment to us. They are pulling through. You have the evidence in your hands, and I have more sitting here for the next issue. I made a commitment before I accepted this job, and hope you feel I am pulling through as well. Now, I want to ask your help in attracting new readership. This is no desperate plea—things are growing quite nicely—but your involvement in *TCJ's* future is important. Don't forget to have anyone you sponsor mention your name. If you are a current subscriber, I'll add another issue onto your own subscription.

And Special Thanks Goes To....

I said *TCJ* is the sum of three parts, the authors, the editor and the readers. There is another group I need to thank as their help and understanding has been of immeasurable value. These are our printers, the folks at the Mill River Press in Brooklyn. Pat Moakley and his gang are another treasure. Have you known someone with "the patience of Job?" That's them! Thanks, guys.

Well, that's it. Let's get on with the show!●

Figure 7 **Interrupt Acknowledge Cycle Timing**



Zilog, from page 6

The System Configuration Register (address EDH) determines the functionality of PIA1 and the Daisy-Chain Configuration (Figure 10). The following list gives you an explanation of each of the eight control bits:

Bit D7 - Reserved and programmed for 0.

Bit D6 - Daisy-chain configuration determines the arrangement of the interrupt priority daisy chain. When this bit is set to 1, priority is as follows:

IEI pin - CTC - SCC - IEO pin

When this bit is 0, priority is as follows:

IEI pin - SCC - CTC - IEO pin

[Bit 6's default (after Reset) is 0].

Bit D5 - Disable /ROMCS. When this bit is set to 1, ROMCS is forced to a 1 regardless of the status of the address decode logic. This bit's default (after Reset) is 0 and /ROMCS function is enabled

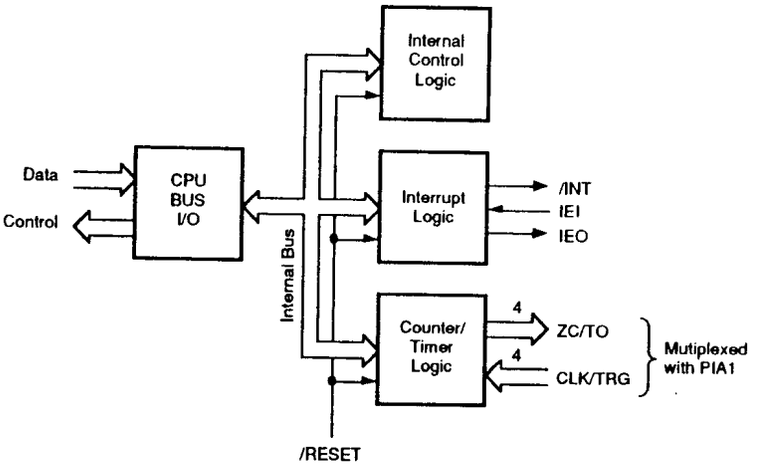
Bits D4-D3 - Reserved and programmed as 00.

Bit D2 - When set to a 1, the Z181 is in ROM Emulator mode. In this mode, bus direction for certain transaction periods are set to the opposite direction to export internal bus transactions outside the SAC. This allows you to use ROM Emulators/Logic Analyzers for applications development. Bit D2's default (after Reset) is 0.

Bit D1 - Reserved and programmed as 0.

Bit D0 - When this bit is set to 1, PIA1 functions as the CTC's I/O pins. Bit D0's default (after Reset) is 0.

Figure 8 **CTC Block Diagram**



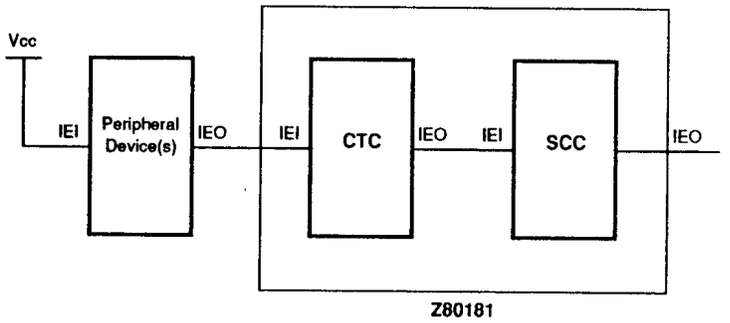


Figure 9A **Peripheral Device as Part of the Daisy Chain**

Directing the Data Bus

When SAC is the bus master, Table 1 shows the state of SAC's data bus. Table 2 shows the state of the SAC data bus when SAC is NOT bus master.

Using the Software

When it's necessary to control on-chip peripherals and features, you can use the SAC's 78 internal registers. Sixty-four of these registers are used by the Z180 MPU control registers. Table 3 lists all the addresses for on-chip I/O control. These control registers are assigned in the SAC's I/O addressing

Figure 9B **Peripheral Device as Part of the Daisy Chain**

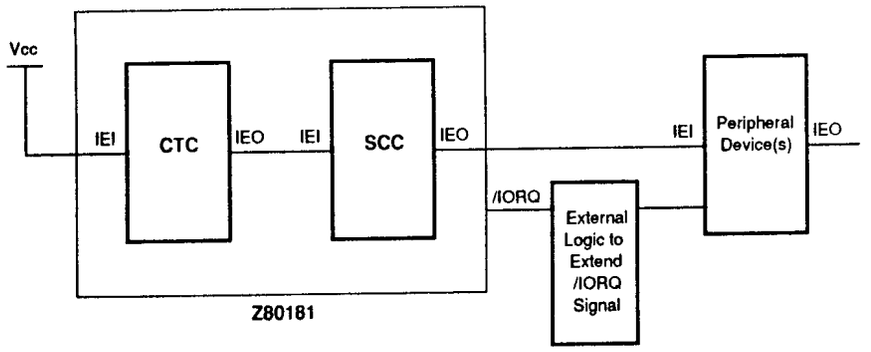
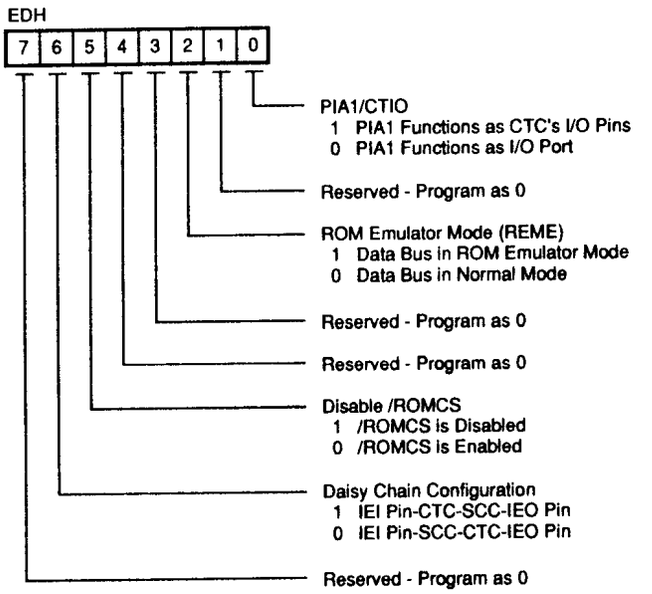


Figure 10 **System Configuration Register**


EDH
7 6 5 4 3 2 1 0
PIA1/CTIO
1 PIA1 Functions as CTC's I/O Pins
0 PIA1 Functions as I/O Port
Reserved - Program as 0
ROM Emulator Mode (REME)
1 Data Bus in ROM Emulator Mode
0 Data Bus in Normal Mode
Reserved - Program as 0
Reserved - Program as 0
Disable /ROMCS
1 /ROMCS is Disabled
0 /ROMCS is Enabled
Daisy Chain Configuration
1 IEI Pin-CTC-SCC-IEO Pin
0 IEI Pin-SCC-CTC-IEO Pin
Reserved - Program as 0

I/O And Memory Transactions

	I/O Write To On-Chip Peripherals (SCC/CTC/ PIA1/PIA2)	I/O Read From On-Chip Peripherals (SCC/CTC/ PIA1/PIA2)	I/O Write To Off-Chip Peripheral	I/O Read From Off-Chip Peripheral	Write To Memory	Read From Memory	Refresh	Z80181 Idle Mode
Z80181 Data Bus (REME Bit = 0)	Out	Z	Out	In	Out	In	Z	Z
Z80181 Data Bus (REME Bit = 1)	Out	Out	Out	In	Out	In	Z	Z

Interrupt Acknowledge Transaction

	Intack For On-chip Peripheral (SCC/CTC)	Intack For Off-chip Peripheral	
Z80181 Data Bus (REME Bit = 0)	Z	In	
Z80181 Data Bus (REME Bit = 1)	Out	In	

Table 1 **Data Bus Direction** (Z181 is Bus Master)

Table 2 **Data Bus Direction for External Bus Master** (Z181 is not Bus Master)

I/O And Memory Transactions

	I/O Read From On-Chip Peripherals (SCC/CTC/ PIA1/PIA2)	I/O Write To On-Chip Peripherals (SCC/CTC/ PIA1/PIA2)	I/O Read From Off-Chip Peripheral	I/O To Off-Chip Peripheral	Write From Memory	Read Memory	Refresh	Ext. Bus-Master Is Idle
Z80181 Data Bus (REME Bit = 0)	In	Out	Z	Z	Z	In	Z	Z
Z80181 Data Bus (REME Bit = 1)	In	Out	Z	Z	Z	In	Z	Z

Interrupt Acknowledge Transaction

	Intack For On-chip Peripheral (SCC/CTC)	Intack For Off-chip Peripheral
Z80181 Data Bus (REME Bit = 0)	Out	In
Z80181 Data Bus (REME Bit = 1)	Out	In

The word "OUT" means that the Z181 data bus direction is in output mode, "IN" means input mode, and "HI-Z" means high impedance.

"REME" stands for "ROM Emulator Mode" and is the status of D2 bit in the System Configuration Register.

space and the I/O addresses are fully decoded from A7-A0 and have no image.

The I/O address for these registers can be relocated in 64 byte boundaries by programming the I/O Control Register at Address xx111111bH. However, do not relocate these registers to addresses from 0C0H since this causes an overlap of the Z180 registers and the 16 Z181 registers (address 0E0H to 0EFH).

As part of the initialization procedure, the OMCR register (Address xx111101bH) is programmed as 0x0xxxxxbH (x: don't care). Program the M1E bit (Bit D7) of the OMCR register to 0 or the interrupt daisy chain is corrupted. To ensure that the timing of the /RD and /IORQ signals are compatible with Z80 peripherals, the /IOC bit (Bit D5) of this register is programmed to 0.

Summary

I have highlighted the main features of the Z181 SAC. Upcoming artiles in *The Computer Journal* will describe an actual Z181 hardware and software design getting into the nitties of a Z181 application. Stay with us.●

Address	Register
00h to 3Fh	Z181 MPU Control Registers (Relocatable to 040h-07Fh, or 080h-0BFh)
E0h	PIA1 Data Direction Register (P1DDR)
E1h	PIA1 Data Port (P1DP)
E2h	PIA2 Data Direction Register (P2DDR)
E3h	PIA2 Data Register (P2DP)
E4h	CTC Channel 0 Control Register (CTC0)
E5h	CTC Channel 1 Control Register (CTC1)
E6h	CTC Channel 2 Control Register (CTC2)
E7h	CTC Channel 3 Control Register (CTC3)
E8h	SCC Control Register (SCCCR)
E9h	SCC Data Register (SCCDR)
EAh	RAM Upper Boundary Address Register (RAMUBR)
EBh	RAM Lower Boundary Address Register (RAMLBR)
ECh	ROM Address Boundary Register (ROMBR)
EDh	System Configuration Register (SCR)
EEh	Reserved
EFh	Reserved

Table 3 **I/O Control Register Address**

"Do Something Good for your Heart... ...Take a Walk"

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Special Close Out Sale on these back issues only.

3 or more, $1.50 each postpaid in the US or $3.00 postpaid airmail outside US.

Issue Number 1:
- RS-232 Interface Part 1
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an 'Epram'

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part 2
- Build Hardware Print Spooler Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler, Part 3
- Beginner's Column: Power Supply Design

Issue Number 5:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board, Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1

Issue Number 19:
- Using the Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part 7
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking & Windows with CP/M: A Review of MTBASIC

Issue Number 20:
- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures & Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC

Issue Number 22:
- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column

Issue Number 23:
- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- The Ampro Little Board Column

Issue Number 24:
- Selecting & Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assemble Code for CP/M
- The C Column: Software Text Filters
- Ampro 186 Column: Installing MS-DOS Software
- The Z-Column
- NEW-DOS: The CCP Internal Commands
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Issue Number 25:
- Repairing & Modifying Printed Circuits
- Z-Com vs. Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro LB
- Building a SCSI Adapter
- NEW-DOS: CCP Internal Commands
- Ampro 186 Networking with SuperDUO
- ZSIG Column

Issue Number 26:
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS

Issue Number 27:
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats

Issue Number 28:
- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control

Issue Number 29:
- Better Software Filter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180: Embedded Processor Design
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- The ZCPR3 Corner

Issue Number 30:
- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- The ZCPR3 Corner
- The CP/M Corner

Issue Number 31:
- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:
- Language Development: Automatic Generation of Parsers for Interactive Systems
- Designing Operating Systems: A ROM based OS for the Z81
- Advanced CP/M: Boosting Performance
- Systematic Elimination of MS-DOS Files: Part 1, Deleting Root Directories & an In-Depth Look at the FCB
- WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII Terminal Based Systems
- K-OS ONE and the SAGE: System Layout and Hardware Configuration
- The ZCPR3 Corner: NZCOM and ZCPR34

Issue Number 33:
- Data File Conversion: Writing a Filter to Convert Foreign File Formats
- Advanced CP/M: ZCPR3PLUS & How to Write Self Relocating Code
- DataBase: The First in a Series on Data Bases and Information Processing
- SCSI for the S-100 Bus: Another Example of SCSI's Versatility
- A Mouse on any Hardware: Implementing the Mouse on a Z80 System
- Systematic Elimination of MS-DOS Files: Part 2, Subdirectories & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:
- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
- New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
- Advanced CP/M: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.
- The Computer Corner

Issue Number 35:
- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner project for S-100 hardware hackers.
- Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
- REL-Style Assembly Language for CP/M and Z-System. Part 1: Selecting your assembler, linker and debugger.
- The Computer Corner

Issue Number 36:
- Information Engineering: Introduction.
- Modula-2: A list of reference books.
- Temperature Measurement & Control: Agricultural computer application.
- ZCPR3 Corner: Z-Nodes, Z-Plan, Amstrand computer, and ZFILE.
- Real Computing: NS32032 hardware for experimenter, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.
- The Computer Corner.

Issue Number 37:
- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.
- The Computer Corner.

Issue Number 38:
- C Math: Handling Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- The Z-System Corner: Shells and ZEX, new Z-Node Central, system security under Z-Systems.
- Information Engineering: The portable Information Age.
- Computer Aided Publishing: Introduction to publishing and Desk Top Publishing.
- Shells: ZEX and hard disk backups.
- Real Computing: The National Semiconductor NS320XX.
- ZSDOS: Anatomy of an Operating System, Part 2.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Issue Number 39:

- Programming for Performance: Assembly Language techniques.
- Computer Aided Publishing: The Hewlett Packard LaserJet.
- The Z-System Corner: System enhancements with NZCOM.
- Generating LaserJet Fonts: A review of Digi-Fonts.
- Advanced CP/M: Making old programs Z-System aware.
- C Pointers, Arrays & Structures Made Easier: Part 3: Structures.
- Shells: Using ARUNZ alias with ZCAL.
- Real Computing: The National Semiconductor NS320XX.
- The Computer Corner.

Issue Number 40:

- Programming the LaserJet: Using the escape codes.
- Beginning Forth Column: Introduction.
- Advanced Forth Column: Variant Records and Modules.
- LINKPRL: Generating the bit maps for PRL files from a REL file.
- WordTech's dBXL: Writing your own custom designed business program.
- Advanced CP/M: ZEX 5.0-The machine and the language.
- Programming for Performance: Assembly language techniques.
- Programming Input/Output With C: Keyboard and screen functions.
- The Z-System Corner: Remote access systems and BDS C.
- Real Computing: The NS320XX
- The Computer Corner.

Issue Number 41:

- Forth Column: ADTs, Object Oriented Concepts.
- Improving the Ampro LB: Overcoming the 88Mb hard drive limit.
- How to add Data Structures in Forth
- Advanced CP/M: CP/M is hacker's haven, and Z-System Command Scheduler.
- The Z-System Corner: Extended Multiple Command Line, and aliases.
- Programming disk and printer functions with C.
- LINKPRL: Making RSXes easy.
- SCOPY: Copying a series of unrelated files.
- The Computer Corner.

Issue Number 42:

- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Screen Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
- Real Computing: The NS 32000.
- The Computer Corner

Issue Number 43:

- Standardize Your Floppy Disk Drives.
- A New History Shell for ZSystem.
- Heath's HDOS, Then and Now.
- The ZSystem Corner: Software update service, and customizing NZCOM.
- Graphics Programming With C: Graphics routines for the IBM PC, and the Turbo C graphics library.
- Lazy Evaluation: End the evaluation as soon as the result is known.
- S-100: There's still life in the old bus.
- Advanced CP/M: Passing parameters, and complex error recovery.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 44:

- Animation with Turbo C Part 1: The Basic Tools.
- Multitasking in Forth: New Micros F68FC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
- DosDisk: MS-DOS disk format emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Real Computing: The NS32000.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.
- The Computer Corner

Issue Number 45:

- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- The Z-System Corner: Using scripts with MEX.
- The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
- Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jetfind.
- Animation with Turbo C: Part 2, screen interactions.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 46:

- Build a Long Distance Printer Driver.
- Using the 8031's built-in UART for serial communications.
- Foundational Modules in Modula 2.
- The Z-System Corner: Patching The Word Plus spell checker, and the ZMATE macro text editor.
- Animation with Turbo C: Text in the graphics mode.
- Z80 Communications Gateway: Prototyping, Counter/Timers, and using the Z80 CTC.

Issue Number 47:

- Controlling Stepper Motors with the 68HC11F
- Z-System Corner: ZMATE Macro Language
- Using 8031 Interrupts
- T-1: What it is & Why You Need to Know
- ZCPR3 & Modula, Too
- Tips on Using LCDs: Interfacing to the 68HC705
- Real Computing: Debugging, NS32 Multitasking & Distributed Systems
- Long Distance Printer Driver: correction
- ROBO-SOG 90
- The Computer Corner

Issue Number 48:

- Fast Math Using Logarithms
- Forth and Forth Assembler
- Modula-2 and the TCAP
- Adding a Bernoulli Drive to a CP/M Computer (Building a SCSI Interface)
- Review of BDS "Z"
- PMATE/ZMATE Macros
- Real Computing
- Z-System Corner: Patching MEX-Plus and TheWord, Using ZEX
- Z-Best Software
- The Computer Corner

Issue Number 49:

- Computer Network Power Protection
- Floppy Disk Alignment w/RTXEB, Pt. 1
- Motor Control w/F68HC11
- Controlling Home Heating & Lighting, Pt 1
- Getting Started in Assembly Language
- LAN Basics
- PMATE/ZMATE Macros
- Real Computing
- Z-System Corner
- Z-Best Software
- The Computer Corner

	U.S.	Foreign (Surface)	Foreign (Airmail)	Total
Subscriptions				
1year (6 issues)	$18.00	$24.00	$38.00	_____
2 years (12 issues)	$32.00	$44.00	$72.00	_____
Back Issues				
18 thru #43	$3.50 ea.		$5.00 ea.	_____
6 or more	$3.00 ea.		$4.50 ea.	_____
#44 and up	$4.50 ea.		$6.00 ea.	_____
6 or more	$4.00 ea.		$5.50 ea.	_____

Issue #s ordered _____

Subscription Total _____

Back Issues Total _____

Total Enclosed _____

Name _____

Address _____

_____

_____

Payment is accepted by check or money order. Checks must be in US funds, drawn on a US bank. Personal checks within the US are welcome.

The Computer Journal
P.O. Box 12, S. Plainfield, NJ 07080-0012
Phone (908) 755-6186

The Computer Corner

By Bill Kibler

In reviewing the last article, I realized that I had missed doing a "state of the computer industry" review. At least once a year I like to indicate where I think things are going and how they might relate to your work-a-day situations. Here it goes.

State of Computing

This last year has brought lots of changes, mostly from the Microsoft—IBM alliance. That "on again, off again" relationship may have put OS2 into the OEM market. I would disagree with lots of computer people that OS2 is dead. I see OS2 becoming instead a platform for special and vertical market users. Some people don't understand the vertical market concept, so lets elaborate.

My company produces products for a very narrow group of users. In the past they have built everything themselves. Now with the advent of cheap clones and their clients wanting to be able to run various DOS programs as well as our programs, the use of proprietary products does not work. You need to use standard platforms, hopefully with enough power to run your programs. MS-DOS lacks the overall power. Unix is not standard enough (maybe later). Digital Research has a better DOS but lacks marketing ability. So the only current candidate is OS2.

From the users' stand point, they could care less about the actual operating system. Their main needs are DOS compatibility, how well your program runs, and cost. These special users have a large number of programs they wish to run, including yours. From the programmers' view, they want a operating system with enough structure to be able to write a program that solves the problem at hand.

Our programs require lots of memory to do the graphics and special text processing operations. This is almost impossible under regular DOS. If there were a protected mode DOS available, lots of companies besides ourselves would be using it. That leaves OS2 as the only means of providing DOS and protected (or using all the memory your machine has as one memory space) operations.

"X-windows will become the real standard for graphic interfaces"

So my crystal ball says OS2 will not die, but become a platform for many companies to put their product on. The results will be few, if any, of the general public buying OS2 for their homes and especially limited business use, as there are few shrink wrapped products that use OS2. The only inroad in this area is X-windows.

I feel fairly strong that X-windows will, over the next few years, become the real standard for graphic interfaces. I see a general feeling that both IBM and Microsoft are becoming the companies *not* to put your product with. They set standards many years ago, and most of the industry jumped on the band wagon. Now that they are trying to change the standard, the industry does not want to change, no matter how much the two biggies wanted them to. I have noticed over the years how our industry has developed. The story of Microsoft is typical. They did not have any special skill or programming expertise that got them started (ok, MBASIC was good, but not that good). Instead they took someone else's work and made a deal with IBM that got them started. Their product and its upgrades over the years have always left lots to be desired, but for the most part, the users had no alternatives.

What happened, then, was a closed market in which they could do anything and the user had to accept it. What I see happening now is Microsoft being caught up in the inability to meet their customer needs. It is time for people to start turning to other choices. Most people are happy with plain old DOS. What they want is better access to the power and memory available in their box. Deskview has given them that for some time (as well as others). The world is turning graphical and so is Deskview. For considerably less than OS2, you will soon be able to run X-window programs and use all your box's memory.

"Microsoft did not have any special skill or programming expertise that got them started."

Unix gets into this picture because lots of boxes can now run it thanks to the 386 cpu. At the same time, the hardware is getting better for Unix, open standards are forming and X-window-type graphical interfaces are getting their final touches. Unix already handles communications between users, which is more than I can say for the problems in DOS-to-LAN operations. Unix has had many years to iron out their problems with the basic structure and now lots of new programs are on the way to making it as easy to use as simple old DOS. Let's not forget all the DOS window emulation programs that give you standard Unix and DOS at the same time.

Now my crystal ball isn't very specific and I do not want you to get the impression that I am selling Deskview or Unix. I do not use either *now*, but have run them and know other who use the products daily. What I am getting at is where

See Computer Corner, page 53

TCJ The Computer Journal Market Place

For: CP/M Users with a sense of humor

WHILE YOU WERE OUT

Mr Bradley

of: Small Computer Support

Address: 24 East Cedar Street

Newington, CT 06111

[ ] called for you

[x] stopped by

[x] wants to hear from you

Message:

Remember Pieces of 8? It's back, better than ever as Eight Bits & Change, a bimonthly newsletter filled with humor, tutorials, graphics and fine technical articles. Only $15 per year in the U.S. ($18 in Canada and $21 foreign.) Subscribe today! Satisfaction guaranteed!

Advent Kaypro Upgrades

TurboROM. Allows flexible configuration of your entire system, read/write additional formats and more. $35

Hard drive conversion kit. Includes interface, controller, TurboROM, software and manual—Everything needed to install a hard drive except the cable and drive! $175 without clock, $200 with clock.

Personality Decoder Board. Run more than two drives, use quad density drives when used with TurboROM. $25

Limited Stock — Subject to prior sale

Call 916-483-0312 eves/weekends or write Chuck Stafford, 4000 Norris Avenue, Sacramento CA 95821

TCJ The Computer Journal Market Place Advertising for Small Business

Looking for a way to get your message across? Advertise in the Market Place!

First Insertion:	$50
Reinsertions:	$35

Rates include typesetting. Payment must accompany order. Foreign orders paid in US funds drawn on a US bank or international money order. Resetting of ad constitutes a new advertisement at first insertion rate. Camera ready copy from laser printers, photo typesetters, etc., are acceptable. Dot matrix, daisy wheel, typewriter output not accepted. Inquire for rates for larger ads if required. Deadline is eight weeks prior to publication date. Mail to:

The Computer Journal
Market Place
PO Box 12
S. Plainfield NJ 07080-0012 USA

CP/M SOFTWARE

100 page Public Domain Catalog, $8.50 plus $1.50 shipping and handling. New Digital Research CP/M 2.2 manual, $19.95 plus $3.00 shipping and handling. Also, MS/PC-DOS Software. Disk Copying, including AMSTRAD. Send self addressed, stamped envelope for free Flyer, Catalog $1.00

Elliam Associates
Box 2664
Atascadero, CA 93423
805-466-8440

Kenmore ZTime-1 Real Time Clocks

Assembled and Tested with
90 Day Warranty
Includes Software

$79.95

Send check or money order to
Chris McEwen
PO Box 12
South Plainfield, NJ 07080
(allow 4-6 weeks for delivery)

Z-System Software Update Service

Provides Z-System public domain software by mail.

Regular Subscription Service
Z3COM Package of over 1.5 MB of COM files
Z3HELP Package with over 1.3 MB of online documentation
Z-SUS Programmers Pack, 8 disks full
Z-SUS Word Processing Toolkit
And More!

For catalog on disk, send $2.00 ($4.00 outside North America)
and your computer format to:

Sage Microsystems East
1435 Centre Street
Newton Centre MA 02159-2469

SAGE MICROSYSTEMS EAST

Selling & Supporting the Best in 8-Bit Software

- Automatic, Dynamic, Universal Z-Systems
  - Z3PLUS: Z-System for CP/M-Plus computers ($70)
  - NZCOM: Z-System for CP/M-2.2 computers ($70)
  - ZCPR34 Source Code: if you need to customize ($50)
- ZSUS: Z-System Software Update Service, public-domain software distribution service (write for a flyer with full information)
- Plu*Perfect Systems
  - Backgrounder ii: CP/M-2.2 multitasker ($75)
  - ZSDOS/ZDDOS: date-stamping DOS ($75, $60 for ZRDOS owners)
  - ZSDOS Programmer's Manual ($10)
  - DosDisk: MS-DOS disk-format emulator, supports subdirectories and date stamps ($30 standard, $35 XBIOS BSX, $45 kit)
  - JetFind: super fast, extemely flexible text file scanner ($50)
- ZMATE: macro text editor / customizable wordprocessor ($50)
- PCED — the closest thing to ARUNZ and LSH (and more) for MS-DOS ($50)
- BDS C — including special Z-System version ($90)
- Turbo Pascal — with new loose-leaf manual ($60)
- SLR Systems (The Ultimate Assembly Language Tools)
  - Z80 assemblers using Zilog (Z80ASM), Hitachi (SLR180), or Intel (SLRMAC) mnemonics
  - linker: SLRNK
  - TPA-based ($50 each) or virtual-memory (special: $160 each)
- ZMAC — Al Hawley's Z-System macro assembler with linker and librarian ($50 disk, $70 with printed manual)
- NightOwl (advanced telecommunications, CP/M and MS-DOS versions)
  - MEX-Plus: automated modem operation with scripts ($60)
  - MEX-Pack: remote operation, terminal emulation ($100)

Next-day shipping of most products with modem download and support available. Order by phone, mail, or modem. Shipping and handling $3 per order (USA). Check, VISA, or MasterCard. Specify exact disk format.

Sage Microsystems East
1435 Centre St., Newton Centre, MA 02159-2469
Voice: 617-965-3552 (9:00am - 11:30pm)
Modem: 617-965-7259 (pw=DDT) (MABOS on PC-Pursuit)